Interactive Physically Based Simulation — Efficient Higher-Order Elements, Multigrid Approaches and Massively Parallel Data Structures

vom Fachbereich Informatik
der Technischen Universität Darmstadt
genehmigte

DISSERTATION

zur Erlangung des akademischen Grades eines
Doktor-Ingenieurs (Dr.-Ing.) von

Dipl.-Inform Daniel Weber

geboren in Darmstadt, Deutschland

Referenten der Arbeit:	Prof. Dr.-Ing. André Stork Technische Universität Darmstadt Prof. Dr. techn. Dieter W. Fellner Technische Universität Darmstadt Prof. Dr.-Ing. Michael Goesele Technische Universität Darmstadt
Tag der Einreichung:	26. Oktober 2015
Tag der mündlichen Prüfung:	7. Dezember 2015

Darmstädter Dissertationen 2016
D 17

Erklärung zur Dissertation

Hiermit versichere ich die vorliegende Dissertation selbständig nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 26. Oktober 2016 Daniel Weber

Abstract

This thesis covers interactive physically based simulation for applications such as computer games or virtual environments. Interactivity, i.e., the option that a user can influence a system, imposes challenging requirements on the simulation algorithms. A simple way to achieve this goal is to drastically limit the resolution in order to guarantee this low computation time. However, with current methods the number of degrees of freedom will be rather low, which results in a low degree of realism. This is due to the fact that not every detail that is important for realistically representing the physical system can be resolved.

This thesis contributes to interactive physically based simulation by developing novel methods and data structures. These can be associated with the three pillars of this thesis: *more accurate discrete representations*, *efficient methods for linear systems*, and *data structures and methods for massively parallel computing*. The novel approaches are evaluated in two application areas relevant in computer generated animation: simulation of dynamic volumetric deformation and fluid dynamics. The resulting accelerations allow for a higher degree of realism because the number of elements or the resolution can be significantly increased.

Zusammenfassung

Physikalisch-basierte Simulationen sind ein wichtiger Bestandteil in vielen Applikationen in der Computergraphik. Computeranimationen, Computerspiele oder virtuelle Umgebungen wirken deutlich realistischer, wenn physikalische Phänomene wie beispielsweise die Bewegung von Rauch, Flüssigkeiten und die elastische Verformung von deformierbaren Körpern realitätsgetreu nachgeahmt werden können. Um möglichst plausible und natürlich wirkende Bewegungsabläufe zu generieren, ist es von Vorteil, die Physik auf Basis der Kontinuumsmechanik zu modellieren, die durch partielle Differenzialgleichungen beschrieben werden und ebenfalls in den Ingenieurwissenschaften Anwendung finden. Dabei werden Energie-, Impuls- und Massenerhaltung gefordert, und die resultierenden Differenzialgleichungen setzen räumliche und zeitliche Ableitung der gesuchten Größen miteinander in Bezug. Zur Berechnung der Bewegung muss in vielen Fällen das Simulationsgebiet vernetzt und ein numerisches Verfahren zur Approximation der Lösung auf dem Rechengitter angewendet werden. Diese numerischen Verfahren sind im Allgemeinen sehr rechenaufwendig und benötigen lange Zeit, um den Bewegungsablauf zu bestimmen. Die Anzahl der Freiheitsgrade des Rechennetzes hat hierbei meist direkten Einfluss auf die Rechenzeit.

Für einige Applikationen wie beispielsweise Computerspiele oder virtuelle Umgebungen ist Interaktivität in Simulationen von zentraler Bedeutung. Interaktiv in diesem Kontext bedeutet, dass die Berechnungszeit eines einzelnen Simulationsschritts unter einhundert Millisekunden betragen muss, sodass ein Benutzer die Simulation als Animation wahrnimmt und gleichzeitig mit ihr interagieren kann. Solche kurzen Berechnungszeiten können mit aktuellen Methoden meist nur erreicht werden, wenn Rechengitter mit einer sehr geringen Anzahl an Freiheitsgraden verwendet werden. Diese können aber meist nicht die gewünschten physikalischen Effekte ausreichend auflösen.

Ziel dieser Arbeit ist es, Interaktivität in physikalisch-basierten Simulationen zu erreichen und gleichzeitig den Detailgrad zu erhöhen. Simulationen mit hochaufgelösten Rechengittern und diesen zeitlichen Anforderungen sind mit aktuellen Methoden nur schwer zu erreichen. Deshalb werden neuartige Methoden entwickelt, die den drei Säulen dieser Arbeit zugeordnet werden können: *Präzisere diskrete Repräsentationsformen*, *effiziente Methoden für lineare Gleichungssysteme* sowie *Datenstrukturen und Methoden für massiv-paralleles Rechnen*. Diese Methoden werden in zwei relevanten Gebieten der Computeranimation, der Strömungssimulation und der Simulation von deformierbaren Materialien, angewendet und evaluiert. Die neuartigen Ansätze beschleunigen physikalisch-basierte Simulationen signifikant und ermöglichen bei interaktiver Rechenzeit mehr Freiheitsgrade und dadurch deutlich mehr Detailreichtum.

Präzisere diskrete Repräsentationsformen erzielen eine höhere Genauigkeit pro Element im Rechengitter und erlauben damit eine deutlich geringere Netzauflösung bei gleichbleibender Qualität. In diesem Kontext wird untersucht, wie Polynome höherer Ordnung in Bernstein-Bézier-Form für die Simulation von Deformation auf Tetraedernetzen eingesetzt werden können und welchen Einfluss dies auf die Qualität und die Rechengeschwindigkeit hat. Weiterhin wird im Umfeld der Strömungssimulation eine verbesserte Modellierung und genauere Repräsentation von eingebetteter Geometrie in Form von teilgefüllten Zellen auf regulären Gittern entwickelt. Damit kann insbesondere Geometrie, die nicht am regulären Gitter ausgerichtet ist, vom Simulator besser aufgelöst und damit genaueres Strömungsverhalten erreicht werden.

In vielen Fällen wird der größte Teil der Rechenzeit vom Aufbau und der Lösung der großen linearen Gleichungssysteme beansprucht, die bei der numerischen Diskretisierung der partiellen Differenzialgleichungen entstehen. Daher werden in dieser Arbeit *effiziente Methoden für lineare Gleichungssysteme* entwickelt und analysiert, welche diesen zeitaufwendigen Prozess beschleunigen. Zum effizienteren Aufbau von Gleichungssystemen wird eine netzbasierte Methode entwickelt, die aufgrund von Nachbarschaftsinformationen in Tetraedernetzen schneller das lineare Gleichungssystem konstruieren kann. Weiterhin werden auf Basis der *präziseren diskreten Repräsentationsformen* zwei neuartige Mehrgittermethoden entwickelt, die auf einer Hierarchie von polynomialen Repräsentationen bzw. auf Hierarchien von Diskretisierungen mit teilgefüllten Zellen basieren. Die entstehenden Gleichungssystemlöser sind deutlich schneller als konventionelle Methoden und ermöglichen damit höhere Auflösungen bei gleichem Budget für die Rechenzeit.

Weiterhin werden *Datenstrukturen und Methoden für massiv-paralleles Rechnen* entwickelt, die speziell auf eine effiziente parallele Ausführbarkeit ausgerichtet sind. Insbesondere für das effiziente, massiv-parallele Rechnen auf Graphikkarten (graphics processing units – GPUs), d.h. die Ausführung auf Tausenden von Prozessoren sind bestimmte Anforderungen zu berücksichtigen, die hier untersucht werden. Eine Datenstruktur für dünnbesetzte Matrizen, die speziell für Deformationssimulationen auf Tetraedernetzen entworfen wurde, optimiert Matrix-Vektor Multiplikationen und ermöglicht damit eine schnelle Lösung von linearen Gleichungssystemen. Weiterhin werden für ein GPU-basiertes Standardverfahren zum Lösen von linearen Gleichungssystemen kleine GPU-Programme zusammengeführt, um den Overhead der Ausführung zu minimieren. Dies ist insbesondere für interaktive Simulationen vorteilhaft, da der zusätzliche Zeitaufwand einen nicht unerheblichen Teil der Rechenzeit ausmachen kann. Weiterhin wird eine Datenstruktur entwickelt, die für teilgefüllte Zellen in Strömungssimulationsszenarien die Datenhaltung minimiert und damit ebenfalls das Lösen von linearen Gleichungssystemen beschleunigt.

Die Methoden, die in dieser Arbeit entwickelt werden, tragen dazu bei, physikalisch-basierte Simulationen signifikant zu beschleunigen. Dies ermöglicht einen deutlich höheren Detailgrad, der durch eine feinere Netzauflösung bei gleicher Rechenzeit erreicht werden kann. Durch die Kombination von den Methoden aus allen drei Säulen kann die Anzahl der Elemente in interaktiven Simulationen deutlich erhöht werden.

Acknowledgements

This dissertation is a result of my work at the Interactive Engineering Technologies department of the Fraunhofer Institute of Computer Graphics Research IGD. There, I found a perfect research environment and the freedom to develop and to follow my own ideas. Many people in this department and beyond have supported me directly or indirectly in my work on and in finishing this dissertation. In the following, I would like to take the opportunity to thank these people.

First of all, I would like to thank my advisor André Stork, who supported my research and kept me motivated throughout the whole time. His office door was always open to discuss ideas, get valuable inspiration and motivation in these frustrating moments that every researcher faces from time to time. As the head of the department, he made it possible to hire students to supported my projects and research work. I am grateful to have had the opportunity to work with him.

Furthermore, I would like to thank my co-advisor Dieter W. Fellner for the possibility to pursue a doctorate at Fraunhofer IGD. A special thanks goes to Michael Goesele, who assessed this dissertation as additional co-advisor on very short notice. In addition, I would like to thank the members of my committee and especially Jan Bender, who supported me by not only being a co-author on several papers but also pushing me to publish my ideas.

Good work can only emerge in a comfortable environment with nice colleagues. It was a pleasure to work with all former and current co-workers in the department. A special thanks goes to Johannes Mueller-Roemer for co-authoring many papers and especially for proof-reading the dissertation. In addition, I would like to thank Christian Altenhofen, who also co-authored some of my publications. I hope I can support you both as much as you did on your way to your doctorates. Furthermore, I would like to thank Gabriele Knöß for relieving me of many administrative tasks and her commitment to perfectly organizing the department's day to day work.

In addition, there are many students who supported my projects or completed their Bachelor or Master theses under my supervision. These are Kerstin Keller, Patrick Charrier, Tobias Ritter, Sascha Raesch, Tim Grasser, Constantin Müller, Jochen Gast, Markus Schnös, Michelle Scott, Andreas Lassmann, Kevin Daun, Felix Loosmann, Mario Selvaggio and Tim Staschewski.

Furthermore, I would like to thank my parents and my sister for their continuous support. I particularly want mention my father, who paved my way to studying computer science by arousing my interest in physical sciences in my childhood. I wish you could have witnessed my way to the doctorate. I am certain you would have been a great support for me.

Finally, I would like to thank my wife Miriam for her patience especially while writing up this dissertation. She and my daughter Mina had to spend many hours without me, while I was sitting at the desk. I hope that I can catch up some of the time now.

Contents

1. Introduction **1**
- 1.1. Contributions . . . 3
  - 1.1.1. Simulation of dynamic volumetric deformation . . . 3
  - 1.1.2. Fluid simulation . . . 4
  - 1.1.3. Publications . . . 5
- 1.2. Structure of this thesis . . . 6

2. Simulation of dynamic volumetric deformation **7**
- 2.1. Background . . . 10
  - 2.1.1. Continuum mechanics . . . 10
  - 2.1.2. Finite element discretization . . . 21
  - 2.1.3. Time integration . . . 29
  - 2.1.4. Solving sparse linear systems . . . 31
- 2.2. Related work . . . 34
  - 2.2.1. Simulation of dynamic volumetric deformation in computer graphics . . . 34
  - 2.2.2. Higher-order FEs for deformation simulation . . . 38
  - 2.2.3. Solving linear systems . . . 38
  - 2.2.4. GPU-based methods . . . 40
  - 2.2.5. Summary . . . 41
- 2.3. The quadratic BB-form method . . . 42
  - 2.3.1. Introduction . . . 42
  - 2.3.2. Trivariate Bernstein-Bézier-form . . . 43
  - 2.3.3. The quadratic BB-form method . . . 45
  - 2.3.4. Mesh-based matrix assembly and traversal . . . 47
  - 2.3.5. Coupling . . . 49
  - 2.3.6. Results and discussion . . . 50
  - 2.3.7. Summary . . . 55
- 2.4. The cubic BB-form method and p-multigrid algorithms . . . 56
  - 2.4.1. Introduction . . . 56
  - 2.4.2. Higher-order finite element discretization of elasticity . . . 57
  - 2.4.3. The p-multigrid method . . . 60
  - 2.4.4. Polynomial intergrid transfer . . . 61
  - 2.4.5. Algorithm . . . 63
  - 2.4.6. Results . . . 65
  - 2.4.7. Summary . . . 73
- 2.5. GPU data structures and methods for efficient deformation simulation . . . 74
  - 2.5.1. Introduction . . . 74
  - 2.5.2. Programming massively parallel hardware . . . 75

2.5.3. BIN-CSR and BIN-BCSR data structure . . . 77
2.5.4. Simulation applications . . . 83
2.5.5. Results . . . 85
2.5.6. Summary . . . 90
2.6. Contributions and conclusions . . . 92
2.6.1. Contributions . . . 92
2.6.2. Discussion . . . 94

3. Fluid simulation **97**
3.1. Background . . . 99
3.1.1. Continuum mechanics . . . 99
3.1.2. Numerically solving the equations . . . 104
3.2. Related work . . . 112
3.2.1. Fluid simulation for computer graphics . . . 112
3.2.2. Sub-grid accurate methods . . . 117
3.2.3. Multigrid approaches for pressure correction . . . 118
3.2.4. Interactive GPU-based methods . . . 119
3.2.5. Summary . . . 119
3.3. Interactive GPU-based fluid dynamics . . . 121
3.3.1. Introduction . . . 121
3.3.2. Geometry sketching framework . . . 121
3.3.3. Geometric cut cells . . . 122
3.3.4. GPU-based fluid simulation . . . 124
3.3.5. Results . . . 128
3.3.6. Summary . . . 132
3.4. The cut-cell geometric multigrid method . . . 133
3.4.1. Introduction . . . 133
3.4.2. Discretization . . . 134
3.4.3. Multigrid solver . . . 137
3.4.4. Results . . . 139
3.4.5. Discussion . . . 146
3.4.6. Summary . . . 146
3.5. Contributions and conclusions . . . 147
3.5.1. Contributions . . . 147
3.5.2. Discussion . . . 148

4. Conclusions and future work **151**
4.1. Research questions . . . 151
4.2. Conclusions . . . 153
4.3. Future work . . . 156
4.3.1. Higher-order approaches . . . 156
4.3.2. Corotational elasticity . . . 157
4.3.3. Accuracy . . . 157
4.3.4. Further GPU extensions . . . 157
4.3.5. Interactive simulation of coupled fluid and deformation effects . . . 158

A. Notation **159**

A.1. Simulation of dynamic volumetric deformation 160
A.2. Fluid simulation 162

B. List of abbreviations **163**

C. Publications and talks **165**
C.1. Authored and co-authored publications 165
C.2. Talks 166

D. Supervising Activities **167**
D.1. Diploma and Master Theses 167
D.2. Bachelor Theses 167

E. Curriculum Vitae **169**

Bibliography **171**

1. Introduction

Simulating the behavior of physical processes is an integral part of many applications in the area of computer graphics and computer animation. In order to achieve a high degree of realism, physically based modeling, especially using continuum mechanics, is of great interest. Physically based simulations are modeled by partial differential equations (PDEs) that describe how a physical system evolves over time. These equations represent the conservation of energy and momentum by relating spatial and temporal changes of the involved quantities with external forces and boundary conditions. Accurately computing physical behavior has a downside, as discretizing and computing a solution for a PDE is generally a compute-intensive task. Usually, finite element methods (FEM), finite volume methods (FVM) or finite differences (FD) are applied to discretize the PDEs. In general, these methods have a high computational complexity as large systems of equations must be solved in every time step. Even when small problem sizes are chosen, standard algorithms tend to produce prohibitively slow simulations.

Application areas in computer graphics can be roughly divided into two categories with their individual requirements:

- Computer generated animation focuses on *visual plausibility*, i.e., the requirement for physical accuracy is relaxed compared to traditional engineering applications, where the reproduction of the real physical behavior is important. In order to generate special effects for film sequences or to produce computer animated movies, the simulated phenomena should rather look as plausible as possible. Computation speed is not a major issue, as animations are usually computed after an artist specified them.
- Computer games or virtual environments, in contrast, require *interactivity*, which is a challenging requirement for physically based simulation. This demanding condition implies that computation times should not exceed 100 milliseconds in order to provide at least ten updates per second [CRM91]. This requirement stems from the fact that the algorithm must be able to process and react to user input while performing the simulation.

The goal of this thesis is to achieve interactivity for physically based simulation. More specifically, the objective is to increase the ratio between accuracy and computation time. In general, the demands for realistic behavior and short computation times are difficult to fulfill simultaneously. Highly detailed simulations are still not feasible in interactive time, emphasizing the importance for further research in this field. Although a considerable amount of effort has been put into research, only moderately sized resolutions can be simulated while satisfying this time constraint. This challenge is being addressed by contributions associated with the following three pillars of this thesis:

1. *more accurate discrete representations*
2. *efficient methods for linear systems*
3. *data structures and methods for massively parallel computing*

These contributions, which are described in more detail in Section 1.1, provide significant acceleration and higher accuracy. The research is conducted in two domains relevant in computer generated animation: simulation of dynamic volumetric deformation and fluid simulation. Being able to speed up the process of physically based simulation provides benefits for both types of requirements mentioned above. For interactive simulations, an

acceleration results in smoother animation due to higher update rates. Alternatively, the number of elements can be increased allowing for more accuracy without spending more computation time. However, a decrease of computation time is also beneficial for computer generated animations. When new methods are available to considerably speed up the simulation process, the number of scenarios that can be computed in the same time increases. This can lead to an improved quality in terms of exploration, meaning the possibility to test different simulation setups in a shorter period of time. Furthermore, increased efficiency enables to enhance accuracy and richness of detail by allowing for a finer spatial resolution or smaller time steps. Thus, this thesis investigates the following research questions that are classified according to the three pillars and have not yet been addressed by previous research:

1. **More accurate discrete representations:** As described in Section 2.3, higher-order bases are more accurate discrete representations for simulated quantities compared to commonly used linear finite elements (FEs) on tetrahedral meshes. Higher-order FEs in Bernstein-Bézier-form (BB-form) additionally enable to set up simulations without the need for numerical integration. The central research questions is:

   1.1 Are higher-order BB-form polynomials beneficial for improving quality and speed of physically based simulation?

   As described in Section 3.3, regular grids offer a computationally attractive opportunity for efficient simulation. Embedded geometry can be discretized by classifying each cell by its belonging to the simulation domain or not. However, the geometric approximation quality of the regular grid representation is limited due to this binary classification of cells. Here the question is:

   1.2 Can cut cells improve the quality of fluid simulations on regular grids?

2. **Efficient methods for linear systems:** In many simulation applications the most computationally intensive part is the setup and computation of the solution of linear systems of equations as described in Section 2.4 and 3.4. Multigrid algorithms have linear time complexity w.r.t. the number of unknowns and theoretically offer promising runtime behavior, especially for large problem sizes. However, good convergence behavior is difficult to achieve and depends on the construction of the discretization hierarchies. The questions regarding the efficient solution of linear systems are:

   2.1 Can efficient multigrid algorithms be constructed based on polynomial hierarchies?

   2.2 How do cut-cell discretization hierarchies improve the convergence rates of multigrid algorithms?

   The solution of the linear system may not be the only bottleneck, especially when employing an efficient solver or limiting the number of iterations. Then the matrix construction also has to be considered when aiming to accelerate the overall simulation algorithm. Especially for higher-order FE discretizations the research question is:

   2.3 How can matrix assembly be accelerated?

3. **Data structures and methods for massively parallel computing:** An option to speed up computations is to make use of parallelization and distribute the computation among multiple processors. Parallelizing simulation algorithms is not trivial, as variables are usually highly coupled and cannot be computed in isolation. Exploiting the computational power of graphics processing units (GPUs) is promising as this massively parallel processor offers thousands of cores on a single chip. It has the potential to further increase the ratio between accuracy and computation time. However, fully exploiting the theoretical computational performance is especially challenging for GPUs, due to their special constraints. Therefore, the research questions addressed are:

   3.1 What are efficient data structures for simulation on GPUs?

   3.2 How can existing simulation methods be adapted to exploit the computational power of GPUs?

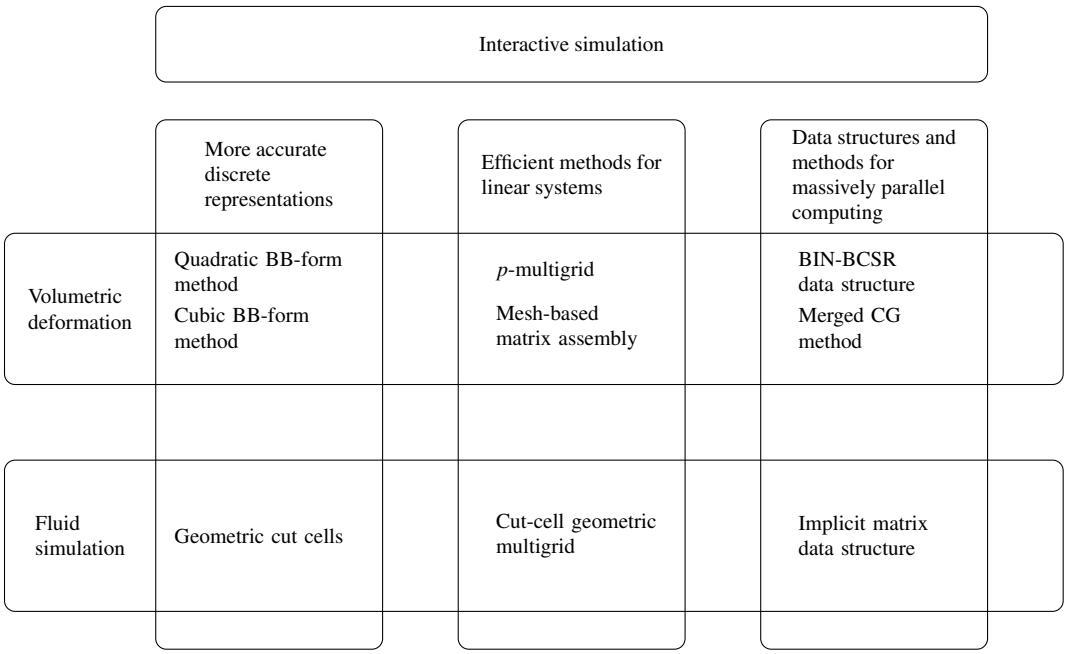


Figure 1.1.: The contributions to address the challenge of interactive simulation are associated with three pillars: *more accurate discrete representations*, *efficient methods for linear systems* and *data structures and methods for massively parallel computing*. These contributions are evaluated in two domains of physically based simulations relevant in computer generated animation, in particular, volumetric deformation and fluid simulation.

1.1. Contributions

This thesis contributes to the field of physically based simulation and addresses the research questions stated in the previous section. The contributions can be associated with the three pillars: *more accurate discrete representations*, *efficient methods for linear systems*, and *data structures and methods for massively parallel computation*. The novel methods will be evaluated in two domains relevant in computer generated animation: dynamic simulation of volumetric deformation and fluid simulation. The contributions in each of the domains are described in the following in more detail. An overview over the classification w.r.t. the three pillars and domains is shown in Fig. 1.1.

1.1.1. Simulation of dynamic volumetric deformation

The state-of-the-art technique in computer graphics for simulating volumetric deformation is to utilize corotational elasticity with linear FEs on tetrahedral meshes and implicit time integration as it is described in Section 2.2. This technique is widespread and requires to solve a large linear system in every time step. A further characteristic is the requirement to locally determine rotations per tetrahedral element. With this information the linear systems must be reconstructed. Therefore, the number of elements and degrees of freedom given by the number of nodes in the tetrahedral mesh determine the runtime. Accurately representing complex geometry

requires a few thousand tetrahedral elements, which can already be too many to guarantee interactivity. The following contributions of this thesis improve the state of the art:

1. In order to address the issue of high computational effort in corotational elasticity, the *quadratic BB-form method* on tetrahedral meshes is developed. This higher-order model requires far fewer elements and results in higher accuracy in comparison to linear FEs. In general, the acceleration is difficult to quantify, as two equivalently accurate discretizations cannot be defined. However, for a specific example with torsional motion an acceleration by a factor of 5 is achieved. Nevertheless, the number of elements can be reduced significantly and the overall computational time is much lower (see Section 2.3.6). Furthermore, the BB-form allows for an easy evaluation of integrals and derivatives and therefore avoids the overhead of numerical integration.
2. Constructing the matrices in every time step for corotational elasticity using higher-order discretizations is a time-intensive operation. The *mesh-based matrix assembly* considers adjacency information in tetrahedral meshes. Using this mesh traversal method accelerates the matrix assembly by 75% for quadratic FEs (see Section 2.3.6).
3. The approach of higher-order simulation is extended to the *cubic BB-form method*. The accuracy is further increased as fewer elements can reproduce complex deformations (see Section 2.4.6).
4. The *p-multigrid algorithm* is developed that is based on BB-form polynomial hierarchies. This method accelerates the solution of linear systems resulting from higher-order discretizations by up to factor of seven (see Section 2.4.6).
5. In order to exploit the high computational power of GPUs, the *BIN-BCSR data structure* is designed to maximize usage of memory bandwidth. This data structure results in an efficient sparse matrix-vector multiplications (SpMV), which is an important operation for solving the corresponding linear system. In comparison to existing GPU data structures, the SpMV operation results in an acceleration of a factor of two to three (see Section 2.5.5). Additionally, the efficient GPU-based matrix construction for corotational elasticity with linear and quadratic FEs results in an overall speedup of a factor of 16 for simulation of dynamic volumetric deformation in comparison to a CPU approach (see Section 2.5.5).
6. Finally, the *merged conjugate gradient method* (MCG) for GPUs is developed in order to reduce the number of required kernel invocations (small GPU programs) in a conjugate gradient (CG) algorithm from nine to three. Due to the constant overhead of kernel launches, the *MCG method* pays off, especially for interactive applications, where the extra time for executing kernels significantly influences the runtime (see Section 2.5.5).

1.1.2. Fluid simulation

Many fluid simulation approaches in computer graphics compute the numerical solution of the incompressible Navier-Stokes or Euler equations as described in Section 3.2. A standard approach is to spatially discretize the equations on a structured, equidistant regular grid and to apply a fractional step method for the time integration. One part which is considered as the major bottleneck in fluid simulations is the pressure correction, where large linear systems must be solved. This is addressed by the following contributions:

1. At first, a GPU-based solution of the linear system for the pressure correction is introduced. An *implicit matrix data structure* is designed, which compactly represents the linear system and reduces the number of required load operations. This data structure results in efficient SpMV operations and improves the runtime w.r.t. standard representations by approximately 20% (see Section 3.3.5).

2. In order to circumvent the usual blocky representation of geometry on regular grids, the *geometric cut-cell method* is developed. This allows for a more detailed representation by explicitly considering the intersections of geometry and the grid resulting in a more accurate computation of flow. Fluid flowing along geometry, which is not aligned with the grid discretization is then computed more accurately (see Section 3.3.5).
3. In order to efficiently solve the large sparse linear systems required for the pressure correction step, the *cut-cell geometric multigrid method* is developed. This method builds on the discretization of cut cells and achieves a more consistent geometric representation on all levels of the multigrid hierarchy. The resulting multigrid algorithm exhibits superior convergence rates for solving the system of linear equations in comparison to existing methods. This drastically reduces the cost of simulation, as the pressure correction step is significantly accelerated by up to a factor of three (see Section 3.4.4). Furthermore, the method is designed to be fully parallelizable so that it can be run efficiently on GPUs resulting in an acceleration by a factor of three to four in comparison to a parallel six core implementation (see Section 3.4.4).

1.1.3. Publications

Parts of this thesis have been already published in journals and conference proceedings:

[WKS*11] WEBER D., KALBE T., STORK A., FELLNER D. W., GOESELE M.: Interactive deformable models with quadratic bases in Bernstein-Bézier-form. *The Visual Computer 27* (2011), 473–483. doi:10.1007/s00371-011-0579-6

[WPnSSF11] WEBER D., PEÑA SERNA S., STORK A., FELLNER D. W.: Rapid CFD for the early conceptual design phase. In *The Integration of CFD into the Product Development Process* (2011), International Association for the Engineering Analysis Community (NAFEMS), NAFEMS, Glasgow, p. 9

[WBS*13] WEBER D., BENDER J., SCHNOES M., STORK A., FELLNER D. W.: Efficient GPU data structures and methods to solve sparse linear systems in dynamics applications. *Computer Graphics Forum 32*, 1 (2013), 16–26. doi:10.1111/j.1467-8659.2012.03227.x

[WMRA*14] WEBER D., MUELLER-ROEMER J., ALTENHOFEN C., STORK A., FELLNER D. W.: A p-multigrid algorithm using cubic finite elements for efficient deformation simulation. In *Virtual Reality Interactions and Physical Simulations (VRIPhys)* (2014), pp. 49–58. doi:10.2312/vriphys.20141223

[WMRSF15] WEBER D., MUELLER-ROEMER J., STORK A., FELLNER D. W.: A cut-cell geometric multigrid poisson solver for fluid simulation. *Computer Graphics Forum (Eurographics proceedings) 34*, 2 (2015), 481–491. doi:10.1111/cgf.12577

[WMRA*15] WEBER D., MUELLER-ROEMER J., ALTENHOFEN C., STORK A., FELLNER D. W.: Deformation simulation using cubic finite elements and efficient *p*-multigrid methods. *Computers & Graphics 53, Part B* (2015), 185 – 195. doi:10.1016/j.cag.2015.06.010

The publication [WMRA*14] was selected as one of three best papers of the conference. An invitation to submit an extended version to the journal Computers & Graphics resulted in the publication [WMRA*15].

Furthermore, this thesis builds upon the diploma thesis

[Web08] WEBER D.: Trivariate Bernstein-Bézier-Techniken für Finite Elemente zur interaktiven Simulation von Deformationen, 2008. Darmstadt, TU, Diplomarbeit, 2008

and extends the approaches described therein. Large parts of the text of the publications were copied verbatim with minor additions and corrections. Furthermore, pictures, tables and diagrams are also reused. The individual sections of this thesis refer to the corresponding publications.

1.2. Structure of this thesis

As this thesis contributes to two domains simulating different physical phenomena and the relevant literature is to a large part disjunct, this document is structured accordingly. Chapter 2 covers the simulation of volumetric deformation. Section 2.1 gives a broad introduction to physically based simulation of volumetric deformation and covers continuum mechanical modeling for elasticity and numerical techniques for solving the associated PDEs. Furthermore, this section defines the terminology of physical and mathematical concepts. In Section 2.2 the related work in this field is reviewed.

In Section 2.3, the *quadratic BB-form method* based on corotational elasticity is presented. Furthermore, the *mesh based-matrix assembly* for quadratic FEs is developed. Section 2.4 extends the higher-order approach to cubic FEs by introducing the *cubic BB-form method*. Furthermore, the *p-multigrid method* based on polynomial hierarchies is developed, which accelerates the solution of linear systems of equations. In Section 2.5 methods and data structures for massively parallel computing are developed that respect the special requirements of GPUs. Therefore, the *BIN-BCSR data structure* is designed that results in efficient SpMV operations. In combination with efficient matrix updates, the number of elements that can be simulated in interactive time is significantly increased. Section 2.6 summarizes the contributions and concludes the topic of volumetric deformation simulation.

Chapter 3 is devoted to the simulation of fluid dynamics. It has a similar structure to the previous chapter. Section 3.1 introduces the equations of motion for fluid simulation and numerical techniques for regular grid discretizations. In Section 3.2, the related work in the field of fluid simulation is reviewed. Section 3.3 introduces a GPU-based fluid simulation technique with the *geometric cut-cell discretization*. Furthermore, the *implicit matrix data structure* is designed. In Section 3.4, the *cut-cell geometric multigrid method* for fluid simulation is introduced. Section 3.5 summarizes the contributions and concludes the topic of fluid simulation.

Section 4 summarizes the contributions and associated research questions. Additionally, the findings regarding physically based simulation are discussed. New open questions are identified and formulated as future work.

A list of all symbols and abbreviations is given in the Appendix A and B.

2. Simulation of dynamic volumetric deformation

Dynamic simulation of volumetric elastic deformation is ubiquitous in the field of computer graphics. The term dynamic simulation refers to computing the deformation of a body over time induced by external forces. Being able to reproduce realistic physical behavior is important and enhances realism in many applications. In applications such as computer games, virtual environments or computer generated animations the deformation is required to be as visually plausible as possible. In addition, the computation has to be efficient, meaning that the time required to determine the motion of a body should be as short as possible. Especially in computer games, there are hard constraints w.r.t. computation times as the state must be updated within milliseconds. This is necessary for guaranteeing interactivity, i.e., to achieve ten or more updates per second [CRM91]. When using haptic devices in virtual environments these constraints are even more challenging, as approximately a thousand updates per second are required in order to give a realistic force feedback experience [BJ05]. In computer generated animation, these requirements are not as strict, as scenarios with predefined forces or collision objects are used and animations are then computed without the need for interaction. However, efficient methods for computing deformation are also desirable as they will lead to shorter iteration times. This makes it possible to compute more scenarios within a given time budget.

An as realistic as possible motion can be obtained by using the physical laws of deformation by means of continuum mechanical modeling described by partial differential equations (PDEs). In contrast to other wide-spread computer animation methods such as position-based dynamics or mass-spring systems, this physical model accurately recreates the characteristics of elastic deformation. Furthermore, physically based simulation converges in the limit of refinement, i.e., with sufficiently small element sizes and time steps the PDEs are solved accurately. In order to compute the effects of elastic deformation, the equations of motion must be discretized w.r.t. time and space. Applying numerical techniques such as the finite element method (FEM) are generally computationally expensive. For an accurate solution, sufficiently small elements sizes are required to ensure that all features of the deformation are spatially resolved. A higher number of elements implies higher computation times as the corresponding linear systems grow. In engineering, the aim is to obtain mesh-independent solutions, meaning that the computed deformation converges when the spatial resolution is increased. However, such an approach is not practical from a computer animation perspective, but it highlights the option to trade accuracy for computation time. Therefore, simulation times can be limited by using a sufficiently low number of elements. Even interactive simulations can be achieved with this strategy, but with standard methods the resulting resolution may be too low to resolve all desired details.

In this thesis or more specifically in this chapter, new approaches for increasing efficiency and accuracy for volumetric deformation simulations are developed. With these novel methods the ratio between accuracy and computation time is increased. In order to achieve the goal of interactive simulation, this thesis contributes novel approaches associated with the three pillars: *more accurate discrete representations*, *efficient methods for linear systems*, and *data structures and methods for massively parallel computing*. These are (also shown in Fig. 2.1 with classification):

- The *quadratic and cubic Bernstein-Bézier-form (BB-form) method* for simulating dynamic volumetric deformation that are a form of higher-order finite elements (FEs) on tetrahedral mesh discretizations.

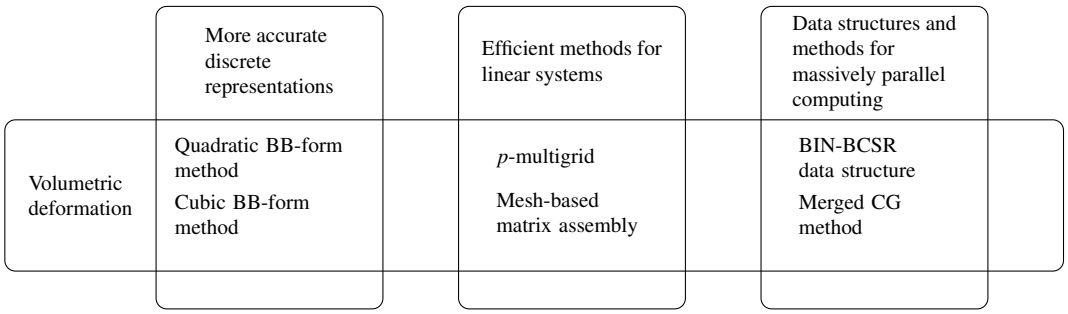


Figure 2.1.: The contributions of this thesis for volumetric deformation simulation are associated with the three pillars *more accurate discrete representations*, *efficient methods for linear systems*, and *data structures and methods for massively parallel computing*.

- The *mesh-based matrix assembly* providing an efficient construction of linear systems for higher-order FEs.
- The *p-multigrid method* that enables efficient solving of systems of linear equations for higher-order BB-form elements. It builds on polynomial hierarchies to compute the solution on different scales, i.e., lower-order polynomials are used as a basis for a more accurate higher-order representation.
- The *BIN-BCSR data structure* for matrices that is specifically designed to respect the special memory access patterns of graphics processing units (GPUs) in order to maximize computational speed.
- The *merged conjugate gradient method* (MCG) reduces the number of execution calls to solve linear systems more quickly on massively parallel hardware.

The higher-order BB-forms are new discrete representations, which increase the accuracy of the simulation and enable the generation of physically plausible motion with significantly fewer elements compared to the standard approach of linear FEs. At the same time, the fidelity of motion increases as higher-order bases are able to better capture details of a deformation, especially for near-incompressible materials. The novel polynomial modeling with the BB-form facilitates the direct evaluation of integrals, differentiation and multiplication in a unified framework. Rotational fields, which are required for state-of-the-art simulations of dynamic volumetric deformation, are approximated. An approximation with piecewise constant polynomials for higher-order FE discretizations significantly reduces the number of required matrix decompositions. The *mesh-based matrix assembly method* provides an efficient way to update linear systems in applications where the matrix changes frequently. Using a tetrahedral mesh data structure directly results in faster construction of linear systems and therefore accelerates their setup. The *p-multigrid method* accelerates linear systems from higher-order discretizations that are at the core of deformation simulations. The polynomial hierarchy represents the solution on different scales and therefore provides a methodology for accelerating the computations similar to standard geometric multigrid solvers. The *BIN-BCSR data structure* is a special matrix representation designed for massively parallel algorithms that is optimized for efficiently computing sparse matrix-vector multiplications (SpMV). Using this data structure for solving the linear systems of equations results in a significant speedup. This further increases the quality of simulations by increasing the maximum number of elements possible at interactive rates. The *MCG method* combines several operations of a conjugate gradient (CG) algorithm so that the number of execution calls is minimized. Especially for interactive simulations, this approach significantly accelerates the solution of linear systems.

This chapter is organized as follows. Section 2.1 introduces the mathematical concepts of modeling deformation and its physical behavior using continuum mechanical approaches. Furthermore, this section defines the terminology and outlines how the resulting PDEs are spatially discretized by adopting the FEM. The description of integration schemes for discretizing the system of ordinary differential equations (ODEs) w.r.t. time and the solution of sparse linear systems finally complete the techniques required to compute the motion of a deformable body. Section 2.2 summarizes the related work in the field of simulation of dynamic volumetric deformation.

In Section 2.3, the novel *quadratic BB-form method* is introduced. Furthermore, the *mesh-based matrix assembly* for quadratic FEs is developed, which enables fast updates for systems of linear equations. Section 2.4 covers the novel *cubic BB-form method* further enhancing accuracy and reducing the number of required elements. In addition, the novel *p-multigrid method* is presented that solves the linear equations based on a polynomial hierarchy resulting in a significant speedup. In Section 2.5, a framework for deformation simulation is introduced that performs massively parallel computations on GPU architectures. This further accelerates the physically based simulation techniques. Therefore, the novel *BIN-BCSR data structure* for sparse matrices is presented, guaranteeing regular access patterns for optimal performance when computing SpMV. Furthermore, the *MCG method* is developed, reducing the number of execution calls and therefore the associated overheads. The resulting speedup factors allow for more degrees of freedom, reduced computation time and higher accuracy. Section 2.6 summarizes the main results for simulation of dynamic volumetric deformation and discusses the findings of the developed methods.

2.1. Background

In this section, the mathematical and physical concepts relevant to the dynamic simulation of deformable bodies are outlined. The techniques described in this thesis focus on dynamic simulation of volumetric deformation. This is in contrast to other approaches that focus on deformation for editing of surface meshes such as presented by Sorkine and Alexa [SA07a]. These methods do not model time-dependent deformation but seek geometric configurations with user-specified constraints instead.

In Section 2.1.1, the equations of equilibrium and motion for dynamic volumetric deformation are derived and stated in terms of PDEs. Section 2.1.2 covers the spatial discretization for turning the PDEs of elasticity into a system of ordinary differential equations (ODEs). Therefore, the FEM is introduced, which is the basis for the methods developed in this chapter. Section 2.1.3 describes an approach that discretizes the ODEs w.r.t. time. This results in linear systems and their solution is covered in Section 2.1.4. Parts of this section are based on the background chapters in the diploma thesis [Web08]. Some of the pictures and sketches are reused.

2.1.1. Continuum mechanics

In the following, the equations of motion for dynamic volumetric elasticity are derived. A more detailed overview of continuum mechanical modeling can be found in the text books by Fung [Fun94] and by Bonet and Wood [BW08].

Continuum mechanical approaches regard the deformable body as a region that covers a subset of the three-dimensional space, the *simulation domain* Ω. The current state of the body, consisting of force, velocity, strain tensor, etc., is modeled as a set of continuous functions on the domain, i.e., it is defined for each point $\mathbf{x} \in \Omega$. The example depicted in Fig. 2.2 shows a body covering the domain Ω with its boundary $\Gamma = \partial\Omega$. As an example two samples of the velocity field $\mathbf{v}(\mathbf{x}) : \Omega \to \mathbb{R}^3$ are shown.

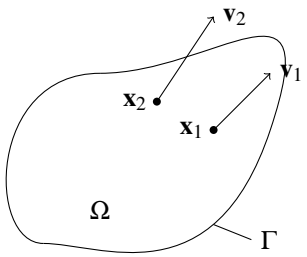


Figure 2.2.: A deformable body covers the simulation domain Ω with its boundary Γ. In a continuum mechanical model the properties and the state of the body are defined by continuous functions on Ω. As an example, two samples of the velocity function $\mathbf{v}(\mathbf{x})$ at positions $\mathbf{x}_1$ and $\mathbf{x}_2$ are sketched.

In order to measure the deformation of an object, its current shape is compared with the undeformed state. Therefore, a body is considered that occupies the domain Ω in the rest state and Ω' in its deformed state as shown in Fig. 2.3. The coordinates and other quantities associated with the two spaces are given in lower and upper case letters, respectively. Lower case letters are used for quantities associated with the rest state Ω and are expressed in *material* or *Eulerian* coordinates. In contrast, upper case letters are associated with the deformed

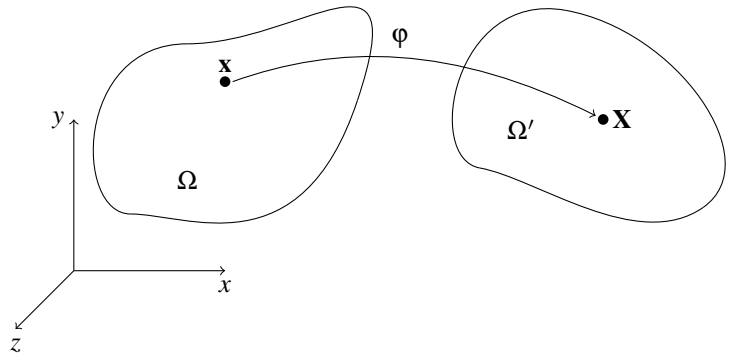


Figure 2.3.: The transformation of a deformable body is given by the deformation map φ that associates points **x** in the rest state Ω to corresponding points **X** in Ω′.

state Ω′ and are given in *spatial* or *Lagrangian* coordinates. The coordinates and its components are then denoted by

$$\mathbf{x} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \in \Omega, \quad \mathbf{X} = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \in \Omega' \tag{2.1}$$

and represent the positions in the corresponding spaces. The mapping between the two spaces is described by the *deformation map* $\varphi : \Omega \to \Omega'$. It associates for each point $\mathbf{x} \in \Omega$ a corresponding point $\mathbf{X} \in \Omega'$

$$\mathbf{X} = \varphi(\mathbf{x}). \tag{2.2}$$

As an alternative representation, a displacement function $\mathbf{u}(\mathbf{x})$ can be defined, as both domains are a subset of the same ambient space $\mathbb{R}^3$. The *displacement map* can then by computed by

$$\mathbf{u}(\mathbf{x}) = \varphi(\mathbf{x}) - \mathbf{x} = \mathbf{X} - \mathbf{x}. \tag{2.3}$$

2.1.1.1. Strain

An elastic body under deformation is subject to internal forces that induce a relative movement. In order to determine elastic forces, the amount of deformation has to be measured and a relationship between the deformation and resulting forces must be set up. The deformation can be measured with a dimensionless quantity called *strain*. As a particularly simple example, a one-dimensional deformation is depicted in Fig. 2.4. There, a bar under load with an initial length l is stretched to a length L. A simple form of strain measurement is to determine the relative change in length, which is given by the so-called *engineering strain*

$$e_E = \frac{L - l}{l} = \frac{\Delta L}{l}. \tag{2.4}$$

It is dimensionless and describes the global deformation of a body with a single scalar value (see Fig. 2.4 (a)). This type of strain measure is very limited as it only represents the global deformation of the body as a whole. In order to get a more precise and spatially varying description of deformation, the change of infinitesimal line

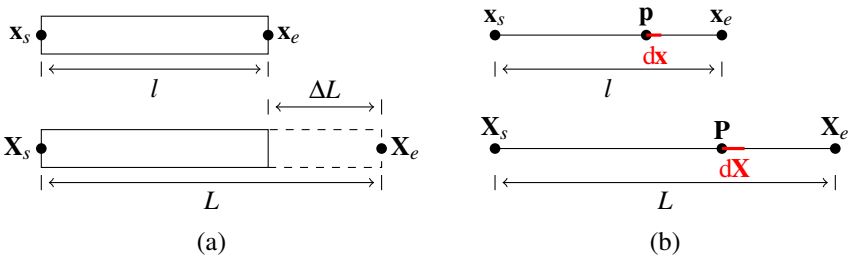


Figure 2.4.: A simple bar with end points $\mathbf{x}_s$ and $\mathbf{x}_e$ is stretched under load. (a) A global strain measure is defined by computing the elongation $\frac{\Delta L}{l}$. (b) A local measure of strain is obtained by considering the change in length of infinitesimal line segments $\mathrm{d}\mathbf{x}$ and $\mathrm{d}\mathbf{X}$.

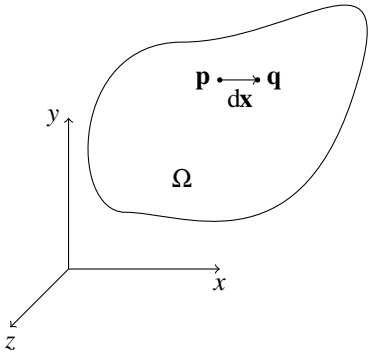


Figure 2.5.: A point $\mathbf{p}$ and a point $\mathbf{q}$ in its direct neighborhood are connected by an infinitesimal line segment $\mathrm{d}\mathbf{x}$.

segments is analyzed. As depicted in Fig. 2.4 (b), the deformation does not only displace the material point $\mathbf{p}$ to $\mathbf{P}$, but it also changes the length of the adjacent line segment $\mathrm{d}\mathbf{x}$. This relative change in length can be expressed by the *infinitesimal engineering strain*

$$\varepsilon_E = \frac{\mathrm{d}\mathbf{x} - \mathrm{d}\mathbf{X}}{\mathrm{d}\mathbf{X}}, \tag{2.5}$$

which is a function potentially varying along the bar.

In order to extend the analysis to three dimensions, it is important to know how infinitesimal line segments are mapped from the rest state to the deformed state. Therefore, consider a point $\mathbf{p}$ and a point $\mathbf{q}$ in the direct neighborhood connected with an infinitesimal line segment $\mathrm{d}\mathbf{x} = \mathbf{q} - \mathbf{p}$ as depicted in Fig. 2.5. The transformation of arbitrary vectors $\mathrm{d}\mathbf{x}$ at a given point into $\mathrm{d}\mathbf{X}$ can be determined by applying the *deformation gradient*. This quantity can be computed by evaluating the gradient of the deformation map φ. It is a linear map and can

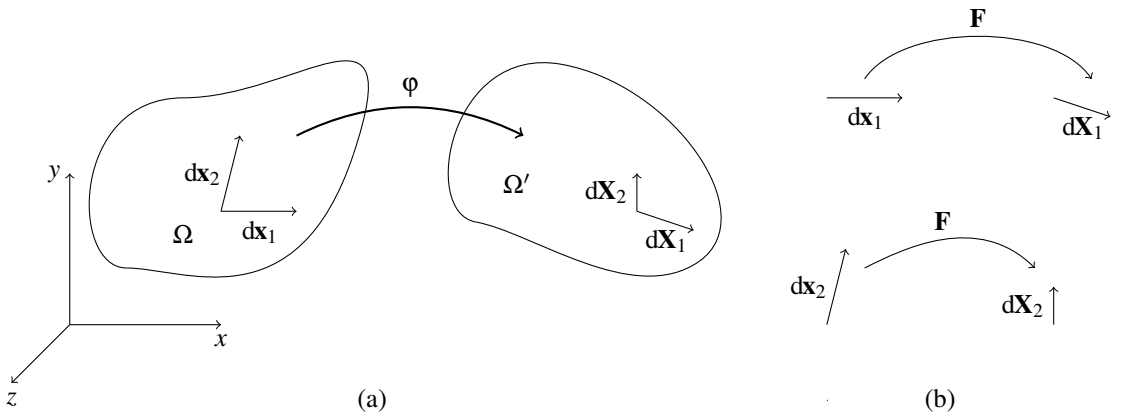


Figure 2.6.: (a) The deformation also affects line segments and changes relative angles and lengths. (b) The transformation of tangent vectors can be evaluated by applying the deformation gradient **F**.

therefore be represented by a 3×3 matrix

$$\mathbf{F} = \nabla \varphi = \begin{pmatrix} \frac{\partial X}{\partial x} & \frac{\partial X}{\partial y} & \frac{\partial X}{\partial z} \\ \frac{\partial Y}{\partial x} & \frac{\partial Y}{\partial y} & \frac{\partial Y}{\partial z} \\ \frac{\partial Z}{\partial x} & \frac{\partial Z}{\partial y} & \frac{\partial Z}{\partial z} \end{pmatrix}. \tag{2.6}$$

Here, ∇ is the *Nabla* operator that applies partial derivatives $(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z})^T$ to a function, which is the gradient or the *Jacobian* of the map in this context. The image $\mathrm{d}\mathbf{X}$ of the vector $\mathrm{d}\mathbf{x}$ can then be determined by a simple matrix multiplication

$$\mathrm{d}\mathbf{X} = \mathbf{F}\,\mathrm{d}\mathbf{x}. \tag{2.7}$$

Computing the transformation corresponds to applying rotational and stretch components. This can be highlighted by applying a pointwise *polar decomposition* of the deformation gradient

$$\mathbf{F} = \mathbf{RS} \tag{2.8}$$

extracting rotational and stretch components. Here, $\mathbf{R}$ is a rotation matrix with determinant $\det(\mathbf{R}) = 1$. The matrix $\mathbf{S}$ is a symmetric matrix containing the stretch components of the deformation gradient. Eq. (2.8) is called a left polar decomposition, the right polar decomposition where the order of two matrices is exchanged also exists, i.e., $\mathbf{F} = \mathbf{S}'\mathbf{R}'$.

Green strain: For a three-dimensional measure of deformation the change in the dot product of two infinitesimal line segments before and after the transformation is analyzed. Therefore, two infinitesimal line segments $\mathrm{d}\mathbf{x}_1$ and $\mathrm{d}\mathbf{x}_2$ are considered as depicted in Fig. 2.6 (a). The transformation of the line segments is given by applying the deformation gradient $\mathrm{d}\mathbf{X}_i = \mathbf{F}\,\mathrm{d}\mathbf{x}_i$ (see Fig. 2.6 (b)). The dot product $\mathrm{d}\mathbf{X}_1 \cdot \mathrm{d}\mathbf{X}_2$ in terms of $\mathrm{d}\mathbf{x}_1$ and $\mathrm{d}\mathbf{x}_2$ can be expressed as

$$\mathrm{d}\mathbf{X}_1 \cdot \mathrm{d}\mathbf{X}_2 = \mathrm{d}\mathbf{X}_1^T\,\mathrm{d}\mathbf{X}_2 = (\mathbf{F}\,\mathrm{d}\mathbf{x}_1)^T\mathbf{F}\,\mathrm{d}\mathbf{x}_2 = \mathrm{d}\mathbf{x}_1^T\mathbf{F}^T\mathbf{F}\,\mathrm{d}\mathbf{x}_2 = \mathrm{d}\mathbf{x}_1^T\mathbf{C}\,\mathrm{d}\mathbf{x}_2. \tag{2.9}$$

Here, the matrix $\mathbf{C} = \mathbf{F}^T\mathbf{F}$ is called *Cauchy-Green deformation tensor*, which is the identity matrix if φ corresponds to a rigid body transformation, where distances within the body are retained. The change in the dot product can then be determined by subtraction

$$d\mathbf{X}_1 \cdot d\mathbf{X}_2 - d\mathbf{x}_1 \cdot d\mathbf{x}_2. \tag{2.10}$$

This can be rearranged to

$$\frac{1}{2}\left(d\mathbf{X}_1 \cdot d\mathbf{X}_2 - d\mathbf{x}_1 \cdot d\mathbf{x}_2\right) = d\mathbf{x}_1 \cdot \varepsilon_G\, d\mathbf{x}_2, \tag{2.11}$$

where the non-linear *Green strain tensor* is then given by

$$\varepsilon_G = \frac{1}{2}\left(\mathbf{C} - \mathbb{I}\right) = \frac{1}{2}\left(\mathbf{F}^T\mathbf{F} - \mathbb{I}\right). \tag{2.12}$$

Here, $\mathbb{I}$ denotes the 3×3 identity matrix. The Green strain tensor is invariant w.r.t. to rigid body movements as only changes in the dot products are measured. This can be verified easily by inserting the polar decomposition Eq. (2.8) into the definition of Green strain

$$\varepsilon_G = \frac{1}{2}\left(\mathbf{F}^T\mathbf{F} - \mathbb{I}\right) = \frac{1}{2}\left(\mathbf{S}^T\mathbf{R}^T\mathbf{R}\mathbf{S} - \mathbb{I}\right) = \frac{1}{2}\left(\mathbf{S}^T\mathbf{S} - \mathbb{I}\right). \tag{2.13}$$

Here, the property $\mathbf{R}\mathbf{R}^T = \mathbf{R}^T\mathbf{R} = \mathbb{I}$ for rotation matrices was applied. Any additional pure rotation then cancels out due to this property.

As an alternative, the Green strain tensor can also be formulated w.r.t. to the gradient of the displacement field. Following Eq. (2.3) the gradient of the displacement map is

$$\nabla\mathbf{u} = \nabla\left(\varphi - \mathbf{x}\right) = \mathbf{F} - \mathbb{I}. \tag{2.14}$$

Rearranging and inserting this into Eq. (2.12) results in

$$\varepsilon_G = \frac{1}{2}\left(\left(\nabla\mathbf{u} + \mathbb{I}\right)^T\left(\nabla\mathbf{u} + \mathbb{I}\right) - \mathbb{I}\right) = \frac{1}{2}\left(\nabla\mathbf{u}^T\nabla\mathbf{u} + \nabla\mathbf{u}^T + \nabla\mathbf{u}\right). \tag{2.15}$$

Cauchy strain: Apparently, the Green strain tensor is a non-linear strain measure as the dependency w.r.t. the displacements or the deformation gradient $\mathbf{F}$, respectively, is quadratic. The linearization of this tensor results in the so-called *Cauchy strain tensor*

$$\varepsilon_C = \frac{1}{2}(\nabla\mathbf{u}^T + \nabla\mathbf{u}) = \frac{1}{2}(\mathbf{F}^T + \mathbf{F}) - \mathbb{I}. \tag{2.16}$$

The Cauchy strain tensor can be interpreted as the symmetric part of the displacement gradient. Decomposing the displacement gradient into a symmetric and skew-symmetric matrix

$$\nabla\mathbf{u} = \varepsilon_C + \omega \tag{2.17}$$

reveals additional properties. Here, ω is a skew symmetric matrix

$$\omega = \frac{1}{2}\left(\nabla\mathbf{u} - \nabla\mathbf{u}^T\right), \tag{2.18}$$

which represents an infinitesimal rotation tensor. Cauchy strain can therefore be interpreted as the result of subtracting infinitesimal rotational motion from the displacement gradient $\varepsilon_C = \nabla\mathbf{u} - \omega$. However, especially for rotational motion the linearization fails to correctly capture the deformation. This can easily be verified by considering a pure rotational motion $\mathbf{R}$, so the polar decomposition Eq. (2.8) of the deformation gradient results in $\mathbf{F} = \mathbf{S}\mathbf{R} = \mathbb{I}\mathbf{R}$. The term $\mathbf{F}^T + \mathbf{F}$ in Eq. (2.16) certainly does not evaluate to the identity matrix and therefore implies non-zero strain, although there is no deformation at all.

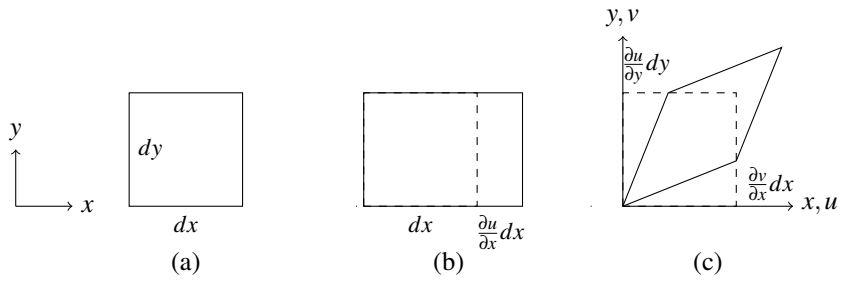


Figure 2.7.: (a) Initial configuration of an undeformed square with side lengths dx and dy, which is deformed by simple stretch (b) and shear (c).

Geometric interpretation of Cauchy strain: The entries in the Cauchy strain tensor for a three-dimensional deformation are given by

$$\varepsilon_C = \begin{pmatrix} \varepsilon_{xx} & \varepsilon_{xy} & \varepsilon_{xz} \\ \varepsilon_{yx} & \varepsilon_{yy} & \varepsilon_{yz} \\ \varepsilon_{zx} & \varepsilon_{zy} & \varepsilon_{zz} \end{pmatrix}. \tag{2.19}$$

In two dimensions the number of entries in the strain tensor reduces to four

$$\varepsilon_C = \begin{pmatrix} \varepsilon_{xx} & \varepsilon_{xy} \\ \varepsilon_{yx} & \varepsilon_{yy} \end{pmatrix} = \frac{1}{2}\left(\nabla \mathbf{u}^T + \nabla \mathbf{u}\right) = \begin{pmatrix} \frac{\partial u}{\partial x} & \frac{1}{2}\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right) \\ \frac{1}{2}\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right) & \frac{\partial v}{\partial y} \end{pmatrix}, \tag{2.20}$$

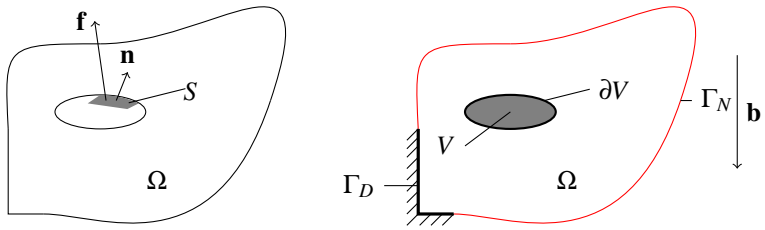
which is a good simplification for analyzing the relationship between the entries and their geometrical meaning. The diagonal entries are given by evaluating the partial derivatives of the displacement field w.r.t. the coordinate axes. Thus, the diagonal elements directly represent the elongation or shortening of the deformable body along the axes. On the other hand, the off-diagonal entries represent the change of angles due to shear motion. Examples for two dimensional deformations are depicted in Fig. 2.7. The deformation in Fig. 2.7 (b) shows an elongation in x-direction, which is comparable to the one-dimensional engineering strain in Fig. 2.4. This is called *normal strain* and results in only one non-zero entry in the strain tensor $\varepsilon_{xx} = \frac{\partial u}{\partial x}$. Fig. 2.7 (c) depicts an angle changing motion with $\varepsilon_{yx} = \varepsilon_{xy} = \frac{1}{2}\left(\frac{\partial u}{\partial y} + \frac{\partial u}{\partial y}\right)$. There, the edges of the square are bent such that the initial right angle is reduced. This motion is called pure *shear strain*. The interpretation can be transferred to the three-dimensional case, where diagonal elements are deformations along the axes and off-diagonal elements represent changes in angles between the coordinate planes, i.e., scale and shear, respectively.

Corotational strain tensor: The *corotational strain* measure is linear in the deformation or displacement gradient and rotationally invariant at the same time. The corotational strain tensor is constructed by evaluating the deformation in a non-rotated coordinate system [HS04]. In order to determine the local rotation, the polar decomposition of the deformation gradient is computed. With the inverse of the stretch component

$$\mathbf{R} = \mathbf{S}^{-1}\mathbf{F}. \tag{2.21}$$

the rotation matrix can be retrieved. The corotational strain is then given by

$$\varepsilon_{CR}(\mathbf{F}) = \varepsilon_C(\mathbf{R}^T\mathbf{F}), \tag{2.22}$$



(a) (b)

Figure 2.8.: (a) The elastic force **f** acting within a deformable body on a surface S can be determined by integrating the traction vector **t** over this surface patch as given in Eq. (2.24). (b) For equilibrium the integral of the traction **t** w.r.t. to the surface ∂V must be in balance with the forces induced by the force density **b**. The state of the deformable body is fully specified by additionally imposing boundary conditions. Here, zero displacement boundary conditions are given on the Dirichlet boundary Γ_D (indicated by the cross-hatched area fixing the body) and homogeneous, i.e., in this case traction-free boundary conditions, on the Neumann boundary Γ_N (marked in red).

where the deformation gradient is rotated back to the initial coordinate system. This strain measure provides rotational invariance by definition and is obviously invariant under general rigid body transformations.

In the following sections, the index of the strain tensor ε is omitted to simplify the notation. The choice of strain tensor, i.e., Cauchy, Green or corotational strain, is apparent from the context. For all cases, strain is a continuous measure, i.e., it is a function $\varepsilon(\mathbf{x})$ that is defined on the simulation domain Ω. It locally describes the amount of deformation of a body. In the case of elasticity, the resulting forces are then computed based on this measure.

2.1.1.2. Stress

The concept of stress is one of the central points in solid mechanics. It is a compact representation to describe the internal forces within the body. Like the strain tensor, the stress tensor σ is given as a continuous function over the simulation domain. In order to determine the elastic forces, the concept of *traction* is required, which represents a force density, i.e., force per unit area. Given a small surface patch and its surface normal **n** within the deformable body, the traction **t** can be determined by a simple matrix multiplication with the stress tensor

$$\mathbf{t} = \sigma \mathbf{n}. \tag{2.23}$$

As depicted in Fig. 2.8 (a), the force **f** that is acting on a surface S can be obtained by integrating the traction **t** over the surface by evaluating the integral

$$\mathbf{f} = \int_S \mathbf{t}\,\mathrm{d}A = \int_S \sigma \mathbf{n}\,\mathrm{d}A, \tag{2.24}$$

where dA is an infinitesimal surface element.

The stress tensor is now analyzed by considering an infinitesimally small volume element, as depicted in Fig. 2.9. There, infinitesimal surface patches dS_i are shown, which are all orthogonal to each other. As the

volume element is infinitesimal, two opposing surface patches are equivalent and it is sufficient to consider the three surface elements dS_1, dS_2 and dS_3. The stress state within this element is completely defined by the three traction vectors $\mathbf{t}_i = (\tau_{i1}, \tau_{i2}, \tau_{i3})^T$ that are acting on the surface elements dS_i. By composing these entries into a matrix, the stress tensor

$$\sigma = \begin{pmatrix} \tau_{11} & \tau_{12} & \tau_{13} \\ \tau_{21} & \tau_{22} & \tau_{23} \\ \tau_{31} & \tau_{32} & \tau_{33} \end{pmatrix} = \begin{pmatrix} \sigma_{xx} & \tau_{xy} & \tau_{xz} \\ \tau_{yx} & \sigma_{yy} & \tau_{yz} \\ \tau_{zx} & \tau_{zy} & \sigma_{zz} \end{pmatrix}. \tag{2.25}$$

is obtained. That way tractions acting on surfaces not aligned with the coordinate axes can be computed by using its normal vector $\mathbf{n}$ and evaluating Eq. (2.23). As in the case of the strain tensor, the entries in the matrix are distinguished between diagonal and off-diagonal elements. The components σ_{xx}, σ_{yy} and σ_{zz} are called *normal stresses* as they represent force densities orthogonal to the coordinate axes. The off-diagonal elements τ_{ij} are called *shear stresses* and represent tractions parallel to the surfaces. Because of angular momentum conservation the stress tensor must be symmetric [Fun94].

2.1.1.3. Material laws

The strain tensor locally describes how the body is deformed, whereas stress represents the elastic forces locally. The relation between strain and stress is described by a material law, which can be any function $\sigma(\varepsilon)$ in its general form. If the function is linear, the material is called *Hookean* and the stress-strain relationships can be described by

$$\sigma_{ij} = \sum_{k,l} D_{ijkl} \varepsilon_{kl}, \tag{2.26}$$

where D_{ijkl} is a fourth order tensor with constant entries. In the case of isotropic elasticity, where the material response is independent of the bodies' orientation, two variables are sufficient to completely describe the elastic behavior. The *Young's modulus* E is directly related to the stiffness of the material and indicates the resistance to deform under external forces, i.e., a higher E implies less deformation under the same load. The *Poisson ratio* ν typically takes values between 0 and 0.5 and determines the volumetric effects due to incompressibility. Fig. 2.10 depicts this effect for a bar under one-dimensional load with a non-zero Poisson ratio. For isotropic elasticity the stress-strain relationship can be described more simply by the so-called *Lamé parameters* λ_L and μ_L. These can be computed easily with Young's modulus and the Poisson ratio by

$$\mu_L = \frac{E}{2(1+\nu)} \quad \lambda_L = \frac{\nu E}{(1+\nu)(1-2\nu)}. \tag{2.27}$$

μ_L is called the *shear modulus* and quantifies the materials resistance to shear motion. λ_L is related to the *bulk modulus* K by $K = \lambda_L + \frac{2}{3}\mu_L$, which quantifies the resistance w.r.t. uniform compression. The stress-strain

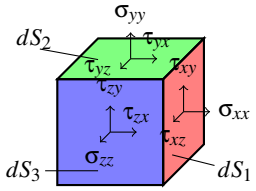


Figure 2.9.: For an infinitesimal volume element, the components of the stress tensor σ can be interpreted as traction vectors acting on infinitesimal surface patches S_i.

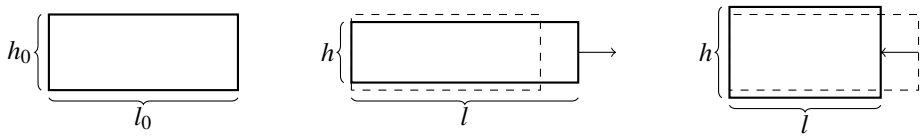


Figure 2.10.: A bar with initial length l_0 and height h_0 is deformed by a one-dimensional load. In addition to the change in length $\Delta l = l - l_0$, the height of the bar changes for a non-zero Poisson ratio $\nu \neq 0$.

relationship for linear isotropic elasticity is then

$$\sigma = 2\mu_L \varepsilon + \lambda_L \operatorname{tr}(\varepsilon)\mathbb{I}, \tag{2.28}$$

where $\mathbb{I}$ is the 3×3 identiy matrix and tr is the trace operator summing up diagonal entries.

For anisotropic materials, more than two independent constants must be chosen in Eq. (2.26) to, e.g., describe the stiffness w.r.t. different coordinate axes. However, the focus in this thesis is on linear isotropic materials, but an extension to linear anisotropy is straightforward. The modeling of non-linear materials requires a more careful derivation. Using linear Cauchy strain or the corotational strain tensor together with a linear material law results in a linear PDE, as derived in the next subsection. A non-linear model, however, will result in a non-linear PDE which requires more effort to solve.

2.1.1.4. Equations of elasticity

A body is in equilibrium if all forces acting on it cancel out on the entire domain. Therefore, consider a closed subvolume V of the body with surface ∂V as shown in Fig. 2.8 (a). If the body is subject to, e.g., gravitational forces $\mathbf{g}$ the equation $\mathbf{f} + \mathbf{g} = 0$ has to be valid pointwise. Turning the gravitational forces into force densities $\mathbf{b}$ and inserting the relation between traction and elastic forces (Eq. (2.24)) results in

$$\oint_S \mathbf{t}\,\mathrm{d}A + \int_V \mathbf{b}\,\mathrm{d}V = 0, \tag{2.29}$$

where $\mathrm{d}A$ or $\mathrm{d}V$ are an infinitesimal surface or volume element, respectively. Inserting Eq. (2.23)

$$\oint_S \sigma\mathbf{n}\,\mathrm{d}A + \int_V \mathbf{b}\,\mathrm{d}V = 0 \tag{2.30}$$

and applying Gauss divergence theorem finally results in

$$\int_V \nabla \cdot \sigma\,\mathrm{d}V + \int_V \mathbf{b}\,\mathrm{d}V = \int_V \nabla \cdot \sigma + \mathbf{b}\,\mathrm{d}V = 0. \tag{2.31}$$

As this relation must be valid for any subvolume V, the integrals can be omitted resulting in the pointwise equilibrium PDE

$$\nabla \cdot \sigma + \mathbf{b} = 0 \tag{2.32}$$

or in component form

$$\begin{pmatrix} \frac{\partial \sigma_x}{\partial x} + \frac{\partial \tau_{yx}}{\partial y} + \frac{\partial \tau_{zx}}{\partial z} \\ \frac{\partial \tau_{xy}}{\partial x} + \frac{\partial \sigma_y}{\partial y} + \frac{\partial \tau_{zy}}{\partial z} \\ \frac{\partial \tau_{xz}}{\partial x} + \frac{\partial \tau_{yz}}{\partial y} + \frac{\partial \sigma_z}{\partial z} \end{pmatrix} + \begin{pmatrix} b_x \\ b_y \\ b_z \end{pmatrix} = 0. \tag{2.33}$$

Additionally, a material law is required that defines how the stress and the strain tensor $\sigma(\varepsilon)$ are related and a definition of strain $\varepsilon(\mathbf{u})$ is required to restate the equation w.r.t. to displacements $\mathbf{u}$. When choosing a linear isotropic material and the linear Cauchy strain tensor, the set of equations is then

$$\begin{aligned} \nabla \cdot \sigma + \mathbf{b} &= 0 \\ \sigma &= 2\mu_L \varepsilon + \lambda_L \operatorname{tr}(\varepsilon)\mathbb{I} \\ \varepsilon &= \frac{1}{2}\left(\nabla \mathbf{u}^T + \nabla \mathbf{u}\right), \end{aligned} \tag{2.34}$$

which are subject to boundary conditions, i.e., prescribed displacements $\mathbf{u}|_{\Gamma_D}$ or tractions $\mathbf{t}|_{\Gamma_N}$ at the Dirichlet boundary Γ_D or Neumann boundary Γ_N, respectively (see Fig. 2.8 (b)). Furthermore, in order to simulate dynamic effects, Eq. (2.32) must be augmented to also take inertial terms into account. Here, the difference of elastic and external forces equates to the change of momentum $\rho\ddot{\mathbf{u}}$ leading to the time dependent PDE

$$\nabla \cdot \sigma + \mathbf{b} = \rho\ddot{\mathbf{u}}, \tag{2.35}$$

where ρ is the density of the material and $\ddot{\mathbf{u}} = \frac{\partial^2 \mathbf{u}}{\partial t^2}$ represents the second derivative of the displacement function w.r.t. time. Here, damping forces are not modeled, but its effect can be easily included later on. The PDE stated in Eq. (2.35) is linear in case of a linear material law and utilizing linear Cauchy strain. It is of second order, as the unknown displacement field is implicitly given via double differentiation, i.e., by the divergence of the stress tensor and the definition of the strain tensor w.r.t. the displacement gradient. Furthermore, it is formulated in the Lagrangian perspective, as the unknown field is specified w.r.t. the initial coordinate system. Eq. (2.35) is given in the so-called *strong form* of elasticity as the equation must hold on the entire simulation domain Ω.

In order to numerically solve the PDE, a transformation into the *weak form* is applied, which is an alternative but equivalent representation such that the equation holds in average over the simulation domain. The detailed and general derivation is described in, e.g., the textbook by Zienckiewicz and Taylor [ZT00], but for completeness the process for linear elasticity is outlined here. The weak form is obtained by multiplying the PDE and the boundary conditions with an arbitrary, vector-valued test function $\mathbf{w}$ and by integrating over the simulation domain Ω or the Neumann boundary Γ_N, respectively

$$\int_\Omega \mathbf{w} \cdot \left[\rho\ddot{\mathbf{u}} - \nabla \cdot \sigma - \mathbf{b}\right] \mathrm{d}V + \int_{\Gamma_N} \mathbf{w} \cdot \left[\sigma\mathbf{n} - \mathbf{t}|_{\Gamma_N}\right] \mathrm{d}A = 0. \tag{2.36}$$

Neumann boundary conditions are prescribed with tractions $\mathbf{t}|_{\Gamma_N} = \sigma\mathbf{n}$, i.e., forces per unit area acting at the boundary. The Dirichlet boundary conditions $\mathbf{u}|_{\Gamma_D}$ can be omitted, as it is possible to restrict the solution space such that the boundary conditions are explicitly satisfied. The test functions $\mathbf{w}$ are arbitrary but are required to vanish on the Dirichlet boundary Γ_D as stated below.

If the original PDE is satisfied, this formulation is stronger as it needs to hold for arbitrary test functions. This transformation into the weak form enables the reduction of the maximum order of derivative in the equation and allows the usage of lower-order bases for the approximation of the unknown function. Whereas the original PDE is of second order, the integration by parts reduces it to first order derivatives. Expanding the terms of the first integral in Eq. (2.36) leads to

$$\int_\Omega \mathbf{w} \cdot \left[\rho\ddot{\mathbf{u}} - \nabla \cdot \sigma - \mathbf{b}\right] \mathrm{d}V = \int_\Omega \rho\mathbf{w} \cdot \ddot{\mathbf{u}} \, \mathrm{d}V - \int_\Omega \mathbf{w} \cdot (\nabla \cdot \sigma) \, \mathrm{d}V - \int_\Omega \mathbf{w} \cdot \mathbf{b} \, \mathrm{d}V. \tag{2.37}$$

The second term $\int_\Omega \mathbf{w} \cdot (\nabla \cdot \sigma) \, \mathrm{d}V$ has to be integrated by parts in order to reduce the order of derivatives. Therefore, the chain rule for the term $\int_\Omega \nabla \cdot (\sigma \cdot \mathbf{w}) \, \mathrm{d}V$ is applied and rearranged

$$\int_\Omega \nabla \cdot (\sigma\mathbf{w}) \, \mathrm{d}V = \int_\Omega \nabla\mathbf{w} : \sigma \, \mathrm{d}V + \int_\Omega \mathbf{w} \cdot (\nabla \cdot \sigma) \, \mathrm{d}V. \tag{2.38}$$

Here, the contraction operator : represents the component-wise multiplication of two tensors, which is

$$\nabla \mathbf{w} : \sigma = \sum_{i,j=0}^{3} (\nabla \mathbf{w})_{ij} \sigma_{ij} \tag{2.39}$$

in this case. As σ is a symmetric tensor, the contraction only affects the symmetric part of $\nabla \mathbf{w}$ and the equation can be rewritten to

$$\nabla \mathbf{w} : \sigma = \frac{1}{2} \left[(\nabla \mathbf{w})^T + \nabla \mathbf{w} \right] : \sigma = \varepsilon(\mathbf{w}) : \sigma. \tag{2.40}$$

In this case, applying the Cauchy strain of the test function $\mathbf{w}$ is equivalent to using the gradient $\nabla \mathbf{w}$. Rearranging Eq. (2.38), inserting Eq. (2.40) and applying Gauss theorem in the second step gives

$$\int_\Omega \mathbf{w} \cdot (\nabla \cdot \sigma) \, \mathrm{d}V = \int_\Omega \nabla \cdot (\sigma \mathbf{w}) \, \mathrm{d}V - \int_\Omega \varepsilon(\mathbf{w}) : \sigma \, \mathrm{d}V = \oint_\Gamma \mathbf{w} \cdot (\sigma \mathbf{n}) \, \mathrm{d}A - \int_\Omega \varepsilon(\mathbf{w}) : \sigma \, \mathrm{d}V, \tag{2.41}$$

where $\Gamma = \Gamma_D \cup \Gamma_N$ is the boundary of the simulation domain, i.e., the union of Dirichlet and Neumann boundaries. Applying this result for Eq. (2.36) leads to

$$\int_\Omega \rho \mathbf{w} \cdot \ddot{\mathbf{u}} \, \mathrm{d}V + \int_\Omega \varepsilon(\mathbf{w}) : \sigma(\mathbf{u}) \, \mathrm{d}V - \int_\Omega \mathbf{w} \cdot \mathbf{b} \, \mathrm{d}V - \oint_\Gamma \mathbf{w} \cdot (\sigma \mathbf{n}) \, \mathrm{d}A + \int_{\Gamma_N} \mathbf{w} \cdot \left[\sigma \mathbf{n} - \mathbf{t}|_{\Gamma_N} \right] \mathrm{d}A = 0, \tag{2.42}$$

where boundary integrals with $\mathbf{w} \cdot \sigma \mathbf{n}$ partially cancel out resulting in

$$\int_\Omega \rho \mathbf{w} \cdot \ddot{\mathbf{u}} \, \mathrm{d}V + \int_\Omega \varepsilon(\mathbf{w}) : \sigma(\mathbf{u}) \, \mathrm{d}V - \int_\Omega \mathbf{w} \cdot \mathbf{b} \, \mathrm{d}V - \int_{\Gamma_N} \mathbf{w} \cdot \mathbf{t}|_{\Gamma_N} \, \mathrm{d}A - \int_{\Gamma_D} \mathbf{w} \cdot \sigma \mathbf{n} \, \mathrm{d}A = 0. \tag{2.43}$$

The last integral represents traction boundary conditions on the Dirichlet boundary. If the test functions $\mathbf{w}$ are restricted to vanish on that boundary as stated above, the integral can be omitted and computations are simplified. True Dirichlet boundary conditions, i.e., prescribed displacements, can be realized by constraining the approximating function. This equation contains only first order derivatives, which are implicitly given by the strain and the stress tensor. Furthermore, it is equivalent to the strong form of elasticity given in Eq. (2.35) but is better suited for a FE discretization.

For the derivation of the discretized equations, it is beneficial to use *Voigt notation* for the strain and stress tensors. In Voigt notation, the symmetry of the tensors is exploited and entries are concatenated into a single vector. This allows to simplify some of the equations using matrix multiplication instead of tensor algebra. Elements in Voigt notation are noted with bold fonts and the stress tensor is then

$$\boldsymbol{\sigma} = (\sigma_{xx}, \sigma_{yy}, \sigma_{zz}, \tau_{xy}, \tau_{xz}, \tau_{yz}) \,. \tag{2.44}$$

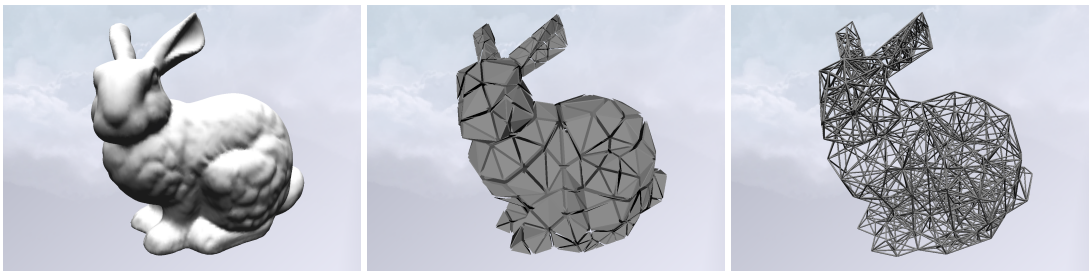
The strain tensor is given by

$$\boldsymbol{\varepsilon} = (\varepsilon_{xx}, \varepsilon_{yy}, \varepsilon_{zz}, \varepsilon_{xy} + \varepsilon_{yx}, \varepsilon_{xz} + \varepsilon_{zx}, \varepsilon_{yz} + \varepsilon_{zy}) \,, \tag{2.45}$$

where in this commonly used notation, the shear components are twice as large as one off-diagonal entry in the tensor formulation. Applying the Voigt notation to the material law $\boldsymbol{\sigma} = \mathbf{D}\boldsymbol{\varepsilon}$ as well, the matrix $\mathbf{D}$ then must account for this modification, where corresponding entries are halved. The contraction $\varepsilon(\mathbf{w}) : \sigma(\mathbf{u})$ can then be rewritten as a scalar product $\boldsymbol{\varepsilon}(\mathbf{w})^T \boldsymbol{\sigma}(\mathbf{u})$. Substituting this and the modified material law into Eq. (2.43), the integral equation is further transformed to

$$\int_\Omega \rho \mathbf{w} \cdot \ddot{\mathbf{u}} \, \mathrm{d}V + \int_\Omega \boldsymbol{\varepsilon}(\mathbf{w})^T \mathbf{D} \boldsymbol{\varepsilon}(\mathbf{u}) \, \mathrm{d}V - \int_\Omega \mathbf{w} \cdot \mathbf{b} \, \mathrm{d}V - \int_{\Gamma_N} \mathbf{w} \cdot \mathbf{t}|_{\Gamma_N} \, \mathrm{d}A = 0, \tag{2.46}$$

which is better suited for deriving the discretized equations described in the next section. Again, it is worth emphasizing that the solution $\mathbf{u}$ must be constructed in such a way that they satisfy the Dirichlet boundary conditions. In contrast, the external forces given as tractions are included in this formulation.



(a) (b) (c)

Figure 2.11.: A tetrahedral discretization is generated from a surface model. (a) Triangular surface model. (b) The resulting tetrahedral discretization is visualized by rendering shrunken tetrahedra. (c) Tetrahedral discretization visualized as a wireframe.

2.1.2. Finite element discretization

In the following, the spatial discretization of the PDE of elasticity is described. Therefore, the FEM is applied to turn the PDE into a set of ODEs, which require discretization w.r.t. time. More details regarding FE modeling can be found in the text books by Bonet and Wood [BW08] and Zienckiewicz and Taylor [ZT00].

The weak form of the elasticity equation (see Eq. (2.46)) derived in the last section as well as the original PDE have an infinite-dimensional solution space, as both are continuous formulations for arbitrary functions defined on Ω. In order to be able to compute a solution or an approximation, this infinite-dimensional space has to be reduced. This can be achieved by discretizing the domain into FEs and setting up local approximations for each element. The per-element formulation is then assembled globally to obtain an aggregated approximation that can then be determined by solving systems of algebraic equations. Starting with a boundary representation given as triangular meshes (see Fig. 2.11 (a)), a volumetric discretization is generated that decomposes and approximates the simulation domain. This can be achieved with appropriate meshing tools such as TetGen [Si15] or CGAL [CGA] that turn the surface description into a tetrahedral mesh as shown in Fig. 2.11 (b) and (c).

2.1.2.1. Piecewise polynomial representation

A tetrahedral mesh $\triangle := (\mathcal{T}, \mathcal{F}, \mathcal{E}, \mathcal{V})$ consists of a set of tetrahedra $\mathcal{T}$, faces $\mathcal{F}$, edges $\mathcal{E}$ and vertices $\mathcal{V}$ and approximates the simulation domain Ω with the union of tetrahedral elements

$$\Omega = \bigcup_{\mathbb{T} \in \mathcal{T}} \mathbb{T}. \tag{2.47}$$

The geometry of each tetrahedron $\mathbb{T} \in \mathcal{T}$ is defined by its corner vertices $\mathbb{T} = [\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3]$. The vertices are oriented counter-clockwise, i.e., $\mathbf{p}_3$ spans a positive volume with the triangle $\mathbf{p}_0, \mathbf{p}_1$ and $\mathbf{p}_2$ and its normal $\mathbf{n} = (\mathbf{p}_1 - \mathbf{p}_0) \times (\mathbf{p}_2 - \mathbf{p}_0)$.

An FE discretization builds a polynomial representation of the unknown field on each tetrahedron. This local view is then assembled to construct a global piecewise polynomial representation of the unknown field. The local polynomial is constructed by requiring that the values at the nodes are interpolated. Therefore, there is a one-to-one correspondence of degrees of freedom and basis functions with these nodes. For complete polynomials on tetrahedral elements, the number of nodes is given by $n = \binom{p+3}{p}$, where p is the polynomial degree. For linear,

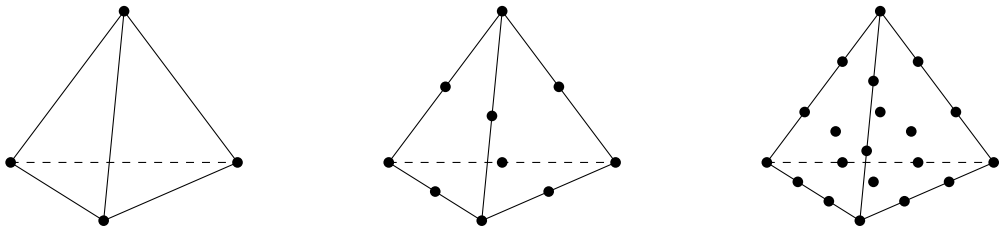


Figure 2.12.: Examples for the node distribution on a tetrahedral FE with linear, quadratic and cubic polynomial degree.

quadratic or cubic FEs, these are 4, 10 or 20 nodes, respectively. In this context, complete means that the basis functions span the space of polynomials $x^i y^j z^k$ with $p = i + j + k$. The geometric positions of the nodes within an element are linear combinations of its vertices. Figure 2.12 shows the node placement for tetrahedral elements with linear, quadratic and cubic polynomials.

For a deformation simulation, the objective is to compute an approximation of the displacement field $\mathbf{u}(\mathbf{x})$ stated in Eq. (2.3). Therefore, a per-element approximation of the displacement field is constructed by the linear combination

$$\mathbf{u}|_{\mathbb{T}}(\mathbf{x}) = \sum_{i=1}^{n} \mathbf{u}_i^{\mathbb{T}} N_i^{\mathbb{T}}(\mathbf{x}). \tag{2.48}$$

Here, $N_i^{\mathbb{T}}(\mathbf{x})$ is a local basis function associated with a node i and $\mathbf{u}_i^{\mathbb{T}}$ is the corresponding nodal displacement. The definition of the basis functions is set up in such a way that they are also non-zero outside the tetrahedral element $\mathbb{T}$. The FE formulation is however designed such that the support is only used within the element. The basis functions $N_i^{\mathbb{T}}(\mathbf{x})$ set up an interpolation and provide a linear combination of the nodal values to locally describe the displacement field in the element $\mathbb{T}$. The linear combination defined in Eq. (2.48) can equivalently be expressed as a matrix multiplication, which is better suited for the upcoming derivation. Therefore, the basis functions are assembled in a $3 \times 3n$ matrix per element $\mathbb{T}$

$$\mathbf{N}_{\mathbb{T}} = \begin{pmatrix} N_1^{\mathbb{T}}(\mathbf{x}) & 0 & 0 & \dots & N_n^{\mathbb{T}}(\mathbf{x}) & 0 & 0 \\ 0 & N_1^{\mathbb{T}}(\mathbf{x}) & 0 & \dots & 0 & N_n^{\mathbb{T}}(\mathbf{x}) & 0 \\ 0 & 0 & N_1^{\mathbb{T}}(\mathbf{x}) & \dots & 0 & 0 & N_n^{\mathbb{T}}(\mathbf{x}) \end{pmatrix}. \tag{2.49}$$

With $\mathbf{N}_{\mathbb{T}}$, Eq. (2.48) can be equivalently written as

$$\mathbf{u}|_{\mathbb{T}}(\mathbf{x}) = \mathbf{N}_{\mathbb{T}} \mathbf{u}_{\mathbb{T}}, \tag{2.50}$$

where $\mathbf{u}_{\mathbb{T}}$ is a $3n \times 1$ vector

$$\mathbf{u}_{\mathbb{T}} = \begin{pmatrix} \mathbf{u}_1^{\mathbb{T}} \\ \vdots \\ \mathbf{u}_n^{\mathbb{T}} \end{pmatrix}, \tag{2.51}$$

which concatenates the n nodal displacements. The discretized and spatially varying deformation gradient can be computed by

$$\mathbf{F}|_{\mathbb{T}} = \mathbb{I} + \nabla \hat{\mathbf{u}}|_{\mathbb{T}} = \mathbb{I} + \sum_{i=1}^{n} \mathbf{u}_i^{\mathbb{T}} \oplus \nabla N_i^{\mathbb{T}}(\mathbf{x}), \tag{2.52}$$

where the gradient of the basis functions $\nabla N_i^{\mathbb{T}}(\mathbf{x}) = \left(\frac{\partial N_i^{\mathbb{T}}}{\partial x} \frac{\partial N_i^{\mathbb{T}}}{\partial y} \frac{\partial N_i^{\mathbb{T}}}{\partial z}\right)$ is required. The term $\mathbf{u}_i^{\mathbb{T}} \oplus \nabla N_i^{\mathbb{T}}(\mathbf{x})$ is the outer product generating a 3×3 matrix from two vectors. The spatially varying Cauchy strain tensor $\boldsymbol{\varepsilon}$ for one element $\mathbb{T}$ can be expressed with the differential matrix

$$\mathbf{B}_{\mathbb{T}} = \begin{pmatrix} \frac{\partial N_1^{\mathbb{T}}}{\partial x} & 0 & 0 & \dots & \frac{\partial N_n^{\mathbb{T}}}{\partial x} & 0 & 0 \\ 0 & \frac{\partial N_1^{\mathbb{T}}}{\partial y} & 0 & \dots & 0 & \frac{\partial N_n^{\mathbb{T}}}{\partial y} & 0 \\ 0 & 0 & \frac{\partial N_1^{\mathbb{T}}}{\partial z} & \dots & 0 & 0 & \frac{\partial N_n^{\mathbb{T}}}{\partial z} \\ \frac{\partial N_1^{\mathbb{T}}}{\partial y} & \frac{\partial N_1^{\mathbb{T}}}{\partial x} & 0 & \dots & \frac{\partial N_n^{\mathbb{T}}}{\partial y} & \frac{\partial N_n^{\mathbb{T}}}{\partial x} & 0 \\ \frac{\partial N_1^{\mathbb{T}}}{\partial z} & 0 & \frac{\partial N_1^{\mathbb{T}}}{\partial x} & \dots & \frac{\partial N_n^{\mathbb{T}}}{\partial z} & 0 & \frac{\partial N_n^{\mathbb{T}}}{\partial x} \\ 0 & \frac{\partial N_1^{\mathbb{T}}}{\partial z} & \frac{\partial N_1^{\mathbb{T}}}{\partial y} & \dots & 0 & \frac{\partial N_n^{\mathbb{T}}}{\partial z} & \frac{\partial N_n^{\mathbb{T}}}{\partial y} \end{pmatrix}. \tag{2.53}$$

It contains the partial derivatives of all basis functions $N_i^{\mathbb{T}}, i = 1, \dots, n$ for the element $\mathbb{T}$. Using this matrix, the Cauchy strain tensor (in Voigt notation) can computed by the matrix multiplication

$$\boldsymbol{\varepsilon}|_{\mathbb{T}} = \mathbf{B}_{\mathbb{T}} \mathbf{u}_{\mathbb{T}}. \tag{2.54}$$

2.1.2.2. Discretization of the elasticity equations

To discretize Eq. (2.46) and to set up a system of algebraic equations a test function must be chosen. In general, there are different approaches for choosing the test functions. A common choice in structural mechanics is to use the same approximation as for the unknown field of displacements

$$\mathbf{w}|_{\mathbb{T}}(\mathbf{x}) = \mathbf{N}_{\mathbb{T}} \mathbf{w}_{\mathbb{T}}. \tag{2.55}$$

This choice and the corresponding discretization are called the *Galerkin approach* and lead to symmetric matrices. A discretization for the accelerations, i.e., the second derivatives of the displacements w.r.t. time, must be provided as well. Applying the same interpolation

$$\ddot{\mathbf{u}}|_{\mathbb{T}}(\mathbf{x}) = \mathbf{N}_{\mathbb{T}} \ddot{\mathbf{u}}_{\mathbb{T}}, \tag{2.56}$$

the weak form of elasticity given in Eq. (2.46) can be discretized. For one element this is

$$\int_{\mathbb{T}} \rho \mathbf{w} \cdot \ddot{\mathbf{u}} \, \mathrm{d}V + \int_{\mathbb{T}} \boldsymbol{\varepsilon}(\mathbf{w})^T \mathbf{D} \boldsymbol{\varepsilon}(\mathbf{u}) \, \mathrm{d}V - \int_{\mathbb{T}} \mathbf{w} \cdot \mathbf{b} \, \mathrm{d}V - \int_{\mathbb{F}} \mathbf{w} \cdot \mathbf{t}|_{\Gamma_N} \, \mathrm{d}A = 0. \tag{2.57}$$

Here, the last term is only evaluated for elements at the boundary, where the integral is evaluated w.r.t. the boundary face $\mathbb{F}$. Inserting the discretizations for displacement, strain, test functions and acceleration given in Eq. (2.50), (2.54), (2.55) and (2.56), respectively, then results in system of ODEs. The first integral of Eq. (2.57) can then be transformed into

$$\int_{\mathbb{T}} \rho \mathbf{w}^T \ddot{\mathbf{u}} \, \mathrm{d}V = \int_{\mathbb{T}} \rho \left(\mathbf{N}_{\mathbb{T}} \mathbf{w}_{\mathbb{T}}\right)^T \mathbf{N}_{\mathbb{T}} \ddot{\mathbf{u}}_{\mathbb{T}} \, \mathrm{d}V = \mathbf{w}_{\mathbb{T}}^T \int_{\mathbb{T}} \rho \mathbf{N}_{\mathbb{T}}^T \mathbf{N}_{\mathbb{T}} \, \mathrm{d}V \ddot{\mathbf{u}}_{\mathbb{T}} = \mathbf{w}_{\mathbb{T}}^T \mathbf{M}_{\mathbb{T}} \ddot{\mathbf{u}}_{\mathbb{T}}. \tag{2.58}$$

The discretization of the second integral results in

$$\int_{\mathbb{T}} \boldsymbol{\varepsilon}(\mathbf{w})^T \mathbf{D} \boldsymbol{\varepsilon}(\mathbf{u}) \, \mathrm{d}V = \int_{\mathbb{T}} \left(\mathbf{B}_{\mathbb{T}} \mathbf{w}_T\right)^T \mathbf{D} \mathbf{B}_{\mathbb{T}} \mathbf{u}_{\mathbb{T}} \, \mathrm{d}V = \mathbf{w}_{\mathbb{T}}^T \int_{\mathbb{T}} \mathbf{B}_{\mathbb{T}}^T \mathbf{D} \mathbf{B}_{\mathbb{T}} \, \mathrm{d}V \mathbf{u}_{\mathbb{T}} = \mathbf{w}_{\mathbb{T}}^T \mathbf{K}_{\mathbb{T}} \mathbf{u}_{\mathbb{T}}. \tag{2.59}$$

The third integral of Eq. (2.57) is discretized by

$$\int_{\mathbb{T}} \mathbf{w}^T \mathbf{b}\,\mathrm{d}V = \int_{\mathbb{T}} (\mathbf{N}_{\mathbb{T}}\mathbf{w}_{\mathbb{T}})^T \mathbf{b}\,\mathrm{d}V = \mathbf{w}_{\mathbb{T}}^T \int_{\mathbb{T}} \mathbf{N}_{\mathbb{T}}^T \mathbf{b}\,\mathrm{d}V = \mathbf{w}_{\mathbb{T}}^T \mathbf{f}_{\mathbb{T}}^{\mathbf{b}}, \tag{2.60}$$

where $\mathbf{f}_{\mathbb{T}}^{\mathbf{b}}$ are nodal forces due to the gravitational force densities $\mathbf{b}$. Additionally, the traction forces that act on the boundary, i.e., the last integral in Eq. (2.57) can be determined in a similar fashion. For nodes at the boundary, the discretized forces $\mathbf{f}_\Gamma$ are added to the nodal force vector

$$\mathbf{f}_{\mathbb{T}} = \mathbf{f}_{\mathbb{T}}^{\mathbf{b}} + \mathbf{f}_\Gamma, \tag{2.61}$$

which contains both types of external forces. Summarizing the above equations, the discretized version of the weak form of elasticity for one element $\mathbb{T}$ is then

$$\mathbf{w}_{\mathbb{T}}^T (\mathbf{M}_{\mathbb{T}}\ddot{\mathbf{u}}_{\mathbb{T}} + \mathbf{K}_{\mathbb{T}}\mathbf{u}_{\mathbb{T}} - \mathbf{f}_{\mathbb{T}}) = 0, \tag{2.62}$$

with mass matrix

$$\mathbf{M}_{\mathbb{T}} = \int_{\mathbb{T}} \mathbf{N}_{\mathbb{T}}^T \rho \mathbf{N}_{\mathbb{T}}\,\mathrm{d}V \tag{2.63}$$

and element's stiffness matrix

$$\mathbf{K}_{\mathbb{T}} = \int_{\mathbb{T}} \mathbf{B}_{\mathbb{T}}^T \mathbf{D}\,\mathbf{B}_{\mathbb{T}}\,\mathrm{d}V. \tag{2.64}$$

In terms of components, the stiffness matrix for linear elasticity can be computed by

$$\left[\mathbf{K}_{\mathbb{T}_{(i,j)}}\right]_{ab} = \int_{\mathbb{T}} \lambda_L \frac{\partial N_i^{\mathbb{T}}}{\partial x_a}\frac{\partial N_j^{\mathbb{T}}}{\partial x_b} + \mu_L \frac{\partial N_i^{\mathbb{T}}}{\partial x_b}\frac{\partial N_j^{\mathbb{T}}}{\partial x_a} + \mu_L \Big(\sum_{c=0}^{2} \frac{\partial N_i^{\mathbb{T}}}{\partial x_c}\frac{\partial N_j^{\mathbb{T}}}{\partial x_c}\Big)\delta_{ab}\,\mathrm{d}V,$$

which can be deduced from the previous equations using the material law with the Lamé parameters given in Eq. (2.28). The indices i, j correspond to the individual nodes of the element and represent 3×3 blocks, which are indexed by a and b. Likewise, the elements of the mass matrix are determined by

$$\left[\mathbf{M}_{\mathbb{T}_{(i,j)}}\right]_{ab} = \int_\Omega \rho \left(N_i^{\mathbb{T}}\right)^T N_j^{\mathbb{T}} \delta_{ab}\,\mathrm{d}V. \tag{2.65}$$

2.1.2.3. Assembly of the global system

The derivation applying the FE discretization leading to Eq. (2.62) has been conducted on a per-element basis. There, local polynomial representations of accelerations, displacements and nodal forces are set in relation to each other to determine the equilibrium within the element. However, the goal is to set up a global equilibrium condition including all m degrees of freedom of the entire tetrahedral mesh that approximates the simulation domain. Therefore, the per-element conditions are superimposed to build a global relation. Taking the displacements as an example, the global representation of the field is given by

$$\mathbf{u}(\mathbf{x}) = \sum_{i=1}^{m} \mathbf{u}_i N_i(\mathbf{x}), \tag{2.66}$$

where the summation is now conducted w.r.t. all m degrees of freedom and basis functions of the entire mesh. Here, the use of $\mathbf{u}_i$ and N_i instead of $\mathbf{u}_i^{\mathbb{T}}$ and $N_i^{\mathbb{T}}$ emphasizes the global dependency of the variables. This definition

extends the local per-element view to a representation that covers the entire simulation domain. In order to evaluate this function w.r.t. to a point $\mathbf{x}$ in the domain, the element $\mathbb{T}$ containing $\mathbf{x}$ must be determined first and then the local interpolation is conducted within the element. The resulting globally defined basis functions have local support, i.e., non-zero entries only occur in elements adjacent to the node they are associated with. Hence, values at nodes influence the corresponding field in all adjacent elements by interpolation.

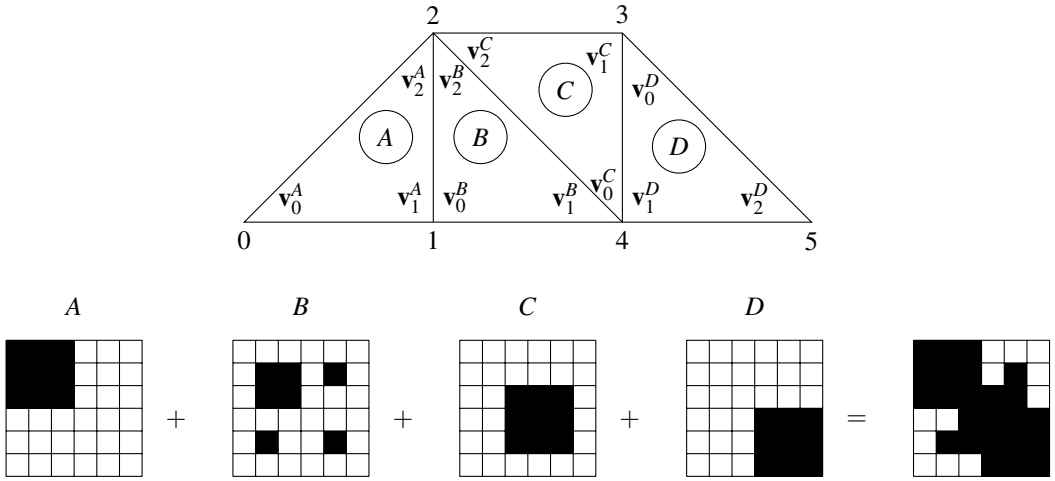


Figure 2.13.: A two-dimensional example of matrix assembly. Top: simple simulation mesh with global and local vertex indices. Bottom: the individual element matrices are assembled into a global matrix. For each of the elements the local stiffness matrix is expanded into a global view and summed up.

Element $\mathbb{T}$	Local index i	Global index $\chi(\mathbb{T},i)$
A	(0,1,2)	(0,1,2)
B	(0,1,2)	(1,4,2)
C	(0,1,2)	(4,3,2)
D	(0,1,2)	(3,4,5)

Table 2.1.: This table represents the local to global mapping $\chi(\mathbb{T},i)$ of node indices for the domain depicted in Fig. 2.13.

The per-element conditions given in Eq. (2.62) can be extended to the entire simulation domain in a similar manner. Therefore, the matrices have to be expanded from the local representation with dimension $\mathbb{R}^{3n\times 3n}$ to the global view $\mathbb{R}^{3m\times 3m}$

$$\mathbf{K}_\mathbb{T} \rightarrow \mathbf{K}'_\mathbb{T}, \mathbf{M}_\mathbb{T} \rightarrow \mathbf{M}'_\mathbb{T}. \tag{2.67}$$

Then the multiplication with the vector of all m accelerations and displacements can be conducted, i.e., the terms $\mathbf{K}'_\mathbb{T}\mathbf{u}$ and $\mathbf{M}'_\mathbb{T}\ddot{\mathbf{u}}$, respectively, can be evaluated. Here, conceptually zeros are filled in for the degrees of freedom that are not adjacent to the element $\mathbb{T}$. By applying this transformation, the global condition of equilibrium can be written as the sum of all element contributions

$$\sum_{\mathbb{T}\in\mathcal{T}} \mathbf{w}_\mathbb{T}^T \left(\mathbf{M}'_\mathbb{T}\ddot{\mathbf{u}} + \mathbf{K}'_\mathbb{T}\mathbf{u} - \mathbf{f}_\mathbb{T}\right) = 0. \tag{2.68}$$

This process is depicted in Fig. 2.13, where an example of matrix assembly is shown for a two-dimensional FE discretization with linear basis functions. A small simulation domain is sketched at the top of the figure and the global matrix construction is depicted below.

Alternatively, the assembly process can be achieved by component-wise summation of corresponding element matrix entries. Additionally, a mapping χ is required that associates the combination of the element $\mathbb{T}$ with its local node i with the corresponding global node k: $\chi(i,\mathbb{T}) \rightarrow k$. Table. 2.1 shows such a mapping for the example domain in Fig 2.13. The global matrices are then constructed by summing over the elements' contribution and adding element entries of local nodes to the corresponding global nodes. An algorithm for this computation is given in Listing 2.1. Here, the map χ is required to determine the position in the global sparse matrix of corresponding local entries.

```
1 def assembly():
2     for 𝕋 in 𝒯:
3         for i in 0,...,n−1:
4             for j in 0,...,n−1:
5                 k = χ(𝕋, i)
6                 l = χ(𝕋, j)
7                 K_kl += [K_𝕋]_ij
```

Listing 2.1.: Algorithm for matrix assembly. The element matrices for each element $\mathbb{T}$ are mapped and added to the corresponding global position.

The assembly process has to be performed for both stiffness matrix and element matrix, which is conceptually equivalent to using the extended matrices $\mathbf{K}'_{\mathbb{T}}$ and $\mathbf{M}'_{\mathbb{T}}$ with additional non-zero entries and compute the sums

$$\mathbf{K} = \sum_{\mathbb{T}} \mathbf{K}'_{\mathbb{T}}, \mathbf{M} = \sum_{\mathbb{T}} \mathbf{M}'_{\mathbb{T}}. \tag{2.69}$$

However, from an implementation perspective this kind of construction is not practical, as filling up large sparse data structures for each element matrix is computationally expensive. Additionally, the vector of all external forces $\mathbf{f}^{\text{ext}}$ is obtained by summing up the individual element contributions $\mathbf{f}_{\mathbb{T}}$. Employing both global matrices and the assembled forces, the spatially discretized version of weak form of elasticity is

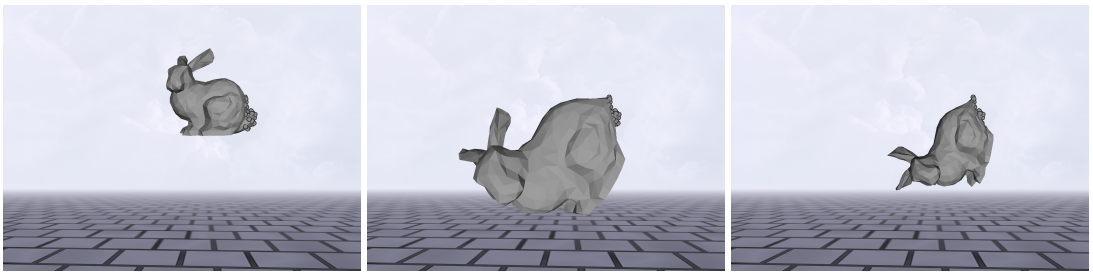
$$\mathbf{w}^T \left(\mathbf{M}\ddot{\mathbf{u}} + \mathbf{K}\mathbf{u} - \mathbf{f}^{\text{ext}} \right) = 0, \tag{2.70}$$

which is equivalent to the sum computed in Eq. (2.68). Here, the local test functions $\mathbf{w}_{\mathbb{T}}$, accelerations $\ddot{\mathbf{u}}_{\mathbb{T}}$ and displacements $\mathbf{u}_{\mathbb{T}}$ are superimposed to construct the global representation.

The continuous test function in the weak formulation (Eq. (2.36)), its local and global discretizations (see Eq. (2.62)) and Eq. (2.70), respectively, were chosen arbitrarily. As the equations must be valid for every choice of test function it is equivalent to requiring that only the right multiplicand in the equation is zero resulting in

$$\mathbf{M}\ddot{\mathbf{u}} + \mathbf{K}\mathbf{u} = \mathbf{f}^{\text{ext}}. \tag{2.71}$$

This is a system of ODEs and represents the equations of motion that describe the global behavior of the deformable body w.r.t. accelerations, forces and displacements at nodes. Here, the dependency on the spatial derivatives has vanished, but it still contains derivatives w.r.t. time, which are resolved by utilizing a time integration scheme as described in Section 2.1.3. The individual terms in Eq. (2.71) correspond to the discretized



(a) (b) (c)

Figure 2.14.: Simulation of a deformable body fixed on the right side and subject to gravitational forces. (a) Initial position of the simulation setup. (b) The resulting deformation shows significant volumetric distortion when using geometrically linear model. (c) Deformation without significant volumetric distortion due to the corotational formulation for the same simulation setup.

versions of the respective forces. $\mathbf{M\ddot{u}}$ are inertial forces, $\mathbf{f}^{\text{el}} = \mathbf{Ku}$ are internal elastic forces and $\mathbf{f}^{\text{ext}}$ are the external forces acting on the body.

For some time integration schemes, applying the concept of mass lumping to the mass matrix is beneficial. In the derived equation, the FE model corresponds to an equal distribution of mass over the elements. This is reflected by the fact that there is a specific non-zero matrix pattern that interpolates mass from adjacent nodes to the inner part of the elements. The process of mass lumping corresponds to concentrating the mass in the nodes. This results in a diagonal matrix $\mathbf{M}_L$, where the values are computed by simply summing up all entries in the respective row. Although computationally attractive, this approach induces artificial damping, which might be undesirable in some simulation applications.

2.1.2.4. Corotational elasticity

The equations derived in the previous section have been conducted with a geometrically linear model, i.e., utilizing Cauchy strain. Although this is a computationally attractive model, as the relationship between displacements and forces is linear, the linear strain measure is only valid for small deformations. In the case of pure compression or stretching along the coordinate axes, this does not generate any undesired visual behavior. However, rotations or rotational deformations are much more problematic as they induce significant volumetric distortion when computing the deformation of the body. Fig. 2.14 shows such behavior, where gravity is applied to a model that is fixed on one side to induce rotational movement. The significant volume growth in the middle picture shows the artifacts caused by geometric linearity in such a scenario. The approach of *corotational elasticity* that is described in this section mitigates this effect [MG04, HS04]. The result based on the same simulation set up is shown in the right picture, where no artifacts are visible.

The idea of the corotational approach is to perform the computation of the elastic forces in the reference coordinate system in order to avoid the artifacts induced by rotational movements. The per-element elastic force vector derived in the previous section can be computed by a matrix multiplication of the stiffness matrix with the displacement vector

$$\mathbf{f}_{\mathbb{T}}^{\text{el}} = {}^{0}\mathbf{K}_{\mathbb{T}}\mathbf{u}_{\mathbb{T}}, \tag{2.72}$$

where the prefixed superscript 0 emphasizes that ${}^0\mathbf{K}_\mathbb{T}$ are the initial element stiffness matrices. Here, the vector of displacements $\mathbf{u}_\mathbb{T} = \mathbf{X}_\mathbb{T} - \mathbf{x}_\mathbb{T}$ corresponds to the difference between the current and the reference positions at the nodes of the element $\mathbb{T}$. Given the local rotation matrix $\mathbf{R}$ that describes relative orientation between the initial and the current coordinate frame, the *corotational force* can be computed by

$$\mathbf{f}_\mathbb{T}^{\text{corot}} = \mathbf{R}\,{}^0\mathbf{K}_\mathbb{T}\left(\mathbf{R}^T\mathbf{X}_\mathbb{T} - \mathbf{x}_\mathbb{T}\right). \tag{2.73}$$

The formula can be interpreted as rotating the current configuration $\mathbf{X}_\mathbb{T}$ back to the initial coordinate system by applying $\mathbf{R}^T\mathbf{X}_\mathbb{T}$. Then the forces are determined in this non-rotated configuration space by ${}^0\mathbf{K}_\mathbb{T}\left(\mathbf{R}^T\mathbf{X}_\mathbb{T} - \mathbf{x}_\mathbb{T}\right)$ and are rotated back into the current coordinate system by applying $\mathbf{R}$. In terms of $\mathbf{u}_\mathbb{T}$, the corotational forces are

$$\mathbf{f}_\mathbb{T}^{\text{corot}} = \mathbf{R}\,{}^0\mathbf{K}_\mathbb{T}\mathbf{R}^T\mathbf{u}_\mathbb{T} - \mathbf{R}\,{}^0\mathbf{K}_\mathbb{T}\left(\mathbf{R}^T\mathbf{x}_\mathbb{T} - \mathbf{x}_\mathbb{T}\right) = \mathbf{K}_\mathbb{T}\mathbf{u}_\mathbb{T} + \mathbf{f}^0. \tag{2.74}$$

The equations of motion given in Eq. (2.71) can be modified to account for the corotational approach by assembling the global force vector $\mathbf{f}^{\text{corot}}$ from $\mathbf{f}_\mathbb{T}^{\text{corot}}$ and by exchanging $\mathbf{f}^{\text{el}}$ with $\mathbf{f}^{\text{corot}}$. The corotational equations of motions are then

$$\mathbf{M}\ddot{\mathbf{u}} + \mathbf{K}\mathbf{u} = \mathbf{f}^{\text{ext}} - \mathbf{f}^0, \tag{2.75}$$

where $\mathbf{K}$ now additionally depends on local element rotations. Therefore, the element stiffness matrix

$$\mathbf{K}_\mathbb{T} = \mathbf{R}\,{}^0\mathbf{K}_\mathbb{T}\mathbf{R}^T \tag{2.76}$$

must be updated and the global matrix $\mathbf{K}$ must be assembled whenever the element rotation matrices $\mathbf{R}$ change. The formulation derived in Eq. (2.75) is equivalent to applying the corotational strain tensor described in Section 2.1.1.1 for measuring the deformation. In contrast to a geometrically linear model, it is additionally rotationally invariant and therefore invariant to rigid body motions.

The computation of the local rotation matrices $\mathbf{R}$ in the discretized FE setting must still be specified. As described in Section 2.1.1.1, the deformation gradient from which the rotational component will be extracted contains both rotational and deformational components. The discretized version of the deformation gradient is already available from Eq. (2.52). Depending on the order p of the discretization, i.e., the order of the basis functions, the deformation gradient is also a spatially varying polynomial of order $p-1$. In the case of linear basis functions with $p = 1$, the deformation gradient is constant within one tetrahedral element and is described by a single matrix. Choosing only the constant representation is a sufficient approximation for extracting the rotation matrix.

First the matrix of the deformation gradient must be determined. Therefore, the vertices of the tetrahedral elements in the reference and deformed configuration are required. In the linear setting, the deformation gradient can be computed by

$$\mathbf{F}|_\mathbb{T} = \mathbb{I} + \sum_{i=1}^{4} \mathbf{u}_i^\mathbb{T} \oplus \nabla N_i^\mathbb{T}(\mathbf{x}) = \sum_{i=1}^{4} \mathbf{X}_i^\mathbb{T} \oplus \nabla N_i^\mathbb{T}(\mathbf{x}). \tag{2.77}$$

Here, the derivatives of the basis functions result from the reference configuration, are constant and can therefore be precomputed. The outer product of basis function gradients $\nabla N_i^\mathbb{T}(\mathbf{x})$ and the vertex positions $\mathbf{X}_i$ are summed up to finally obtain the deformation gradient matrix. The rotation matrix can be extracted by computing the polar decomposition

$$\mathbf{F} = \mathbf{R}\mathbf{S}. \tag{2.78}$$

Therefore, $\mathbf{S}$ needs to be determined and

$$\mathbf{R} = \mathbf{S}^{-1}\mathbf{F} \tag{2.79}$$

is evaluated to obtain the local rotation. $\mathbf{S}$ can be obtained by observing that $\mathbf{F}^T\mathbf{F}$ equals $\mathbf{S}^T\mathbf{S}$ (see, e.g., the work of Irving et al. [ITF04]) by evaluating

$$\mathbf{F}^T\mathbf{F} = (\mathbf{RS})^T\mathbf{RS} = \mathbf{S}^T\mathbf{R}^T\mathbf{RS} = \mathbf{S}^T\mathbf{S}. \tag{2.80}$$

$\mathbf{S}$ can be determined by computing the square root $\sqrt{\mathbf{F}^T\mathbf{F}}$. In order to achieve that, the diagonalization $\mathbf{D}$ of $\mathbf{F}^T\mathbf{F}$ needs to be computed by performing an eigendecomposition

$$\mathbf{F}^T\mathbf{F} = \mathbf{UDU}^T, \tag{2.81}$$

where $\mathbf{U}$ contains the eigenvectors of $\mathbf{F}^T\mathbf{F}$ and is an orthonormal matrix. $\mathbf{D}$ is given by

$$\mathbf{D} = \mathbf{U}^T\mathbf{F}^T\mathbf{FU}. \tag{2.82}$$

$\mathbf{S}$ is then finally obtained by evaluating

$$\mathbf{S} = \mathbf{U}\sqrt{\mathbf{D}}\mathbf{U}^T = \mathbf{U}\begin{pmatrix}\sqrt{D_{11}} & 0 & 0\\ 0 & \sqrt{D_{22}} & 0\\ 0 & 0 & \sqrt{D_{33}}\end{pmatrix}\mathbf{U}^T, \tag{2.83}$$

where the square root is applied to the diagonal entries.

If the tetrahedral element is inverted due to heavy deformation, the determinant of $\mathbf{F}$ will be negative. In this case, the rotation matrix computed in Eq. (2.79) will also have a negative determinant $\det(\mathbf{R}) = -1$, which represents a reflection rather than a rotation. $\det(\mathbf{R})$ is negative, as the diagonalization $\sqrt{\mathbf{D}}$ of $\mathbf{S}$ computed in Eq. (2.83) has positive values by construction. However, a rotation matrix is required in the corotational approach, as otherwise the forces are computed incorrectly. Therefore, the polar decomposition has to be modified whenever $\det(\mathbf{F}) < 0$. This can be achieved by negating one entry in the diagonal matrix $\sqrt{\mathbf{D}}$.

If an element is degenerated to a plane, line or a point, one, two or three eigenvalues, i.e., the entries in the diagonal matrix, are zero. In order to obtain a rotation matrix in these cases, $\mathbf{S}$ is further diagonalized within the polar decomposition

$$\mathbf{F} = \mathbf{RS} = \mathbf{RU}\hat{\mathbf{S}}\mathbf{U}^T = \mathbf{V}\hat{\mathbf{S}}\mathbf{U}^T, \tag{2.84}$$

which is equivalent to a singular value decomposition (SVD) of the deformation gradient [ST08]. If an entry $\hat{\mathbf{S}}$ is zero, the corresponding column in $\mathbf{V}$ is replaced by the cross product of the other two columns to ensure the matrix to be orthonormal. If more than one entry is zero, an orthonormal system is constructed from the columns associated with the non-zero values. In these cases, the rotation can be computed by $\mathbf{R} = \mathbf{VU}^T$, where $\mathbf{V}$ is modified as described above.

2.1.3. Time integration

The equations of motion given in Eq. (2.71) for the geometrically linear model or Eq. (2.75) for the corotational approach are systems of ODEs, which still contain derivatives w.r.t. time. In order to compute a body's motion under the influence of external forces, an appropriate *time integration scheme* must be applied to resolve the time dependency. A time integration scheme computes successive values of the accelerations $\ddot{\mathbf{u}}_{(i)}$, velocities $\dot{\mathbf{u}}_{(i)}$ and displacements $\mathbf{u}_{(i)}$ by discretizing time into steps $t_i, i = 0, \ldots$. Here, the index(i) of the computed variables corresponds to the state at time t_i, e.g., $\mathbf{u}_{(i)} = \mathbf{u}(t_i)$. If the time steps are equidistant, the difference between two consecutive time steps is constant and denoted by $\Delta t = t_{i+1} - t_i$.

The equations of motion can be written as the balance of the inertial force with other forces acting on the body

$$\mathbf{M}\ddot{\mathbf{u}} = -\mathbf{f}^{\text{el}} + \mathbf{f}^{\text{ext}}. \tag{2.85}$$

Here, $\mathbf{f}^{\text{el}}$ is either $\mathbf{Ku}$ for the geometrically linear model or $\mathbf{Ku}+\mathbf{f}^0$ for the corotational approach. The system of differential equations is of dimension $\mathbb{R}^{3m\times 3m}$, where m is the number of nodes in the FE mesh. Rewriting the above equation as

$$\ddot{\mathbf{u}} = \mathbf{M}^{-1}\left(-\mathbf{f}^{\text{el}} + \mathbf{f}^{\text{ext}}\right) = \mathbf{M}^{-1}\mathbf{f}, \tag{2.86}$$

with $\mathbf{f}$ representing the sum of all forces, the second order differential equation is ready for integration. In order to solve the equations, the system is transformed into two first order systems by introducing the velocity variable $\dot{\mathbf{u}}$

$$\frac{d}{dt}\begin{pmatrix}\mathbf{u}\\ \dot{\mathbf{u}}\end{pmatrix} = \begin{pmatrix}\dot{\mathbf{u}}\\ \mathbf{M}^{-1}\mathbf{f}\end{pmatrix}. \tag{2.87}$$

A particularly simple approach is *explicit time integration*

$$\begin{pmatrix}\mathbf{u}_{(i+1)}\\ \dot{\mathbf{u}}_{(i+1)}\end{pmatrix} = \begin{pmatrix}\mathbf{u}_{(i)}\\ \dot{\mathbf{u}}_{(i)}\end{pmatrix} + \Delta t\begin{pmatrix}\dot{\mathbf{u}}_{(i)}\\ \mathbf{M}^{-1}\mathbf{f}_{(i)}\end{pmatrix}, \tag{2.88}$$

which is fast to compute and easy to implement as the update rules only consist of a multiplication and addition operation per degree of freedom. O'Brien and Hodgins [OH99] adopted explicit time integration for simulating of deformation and brittle fracture with tetrahedral FE. Explicit time integration is also used by the space and time adaptive approach proposed by Debunne et al. [DDCB01]. The adaptivity of the time steps is necessary, as the stability of the solution is bounded by the *Courant condition*, which depends on the stiffness of the modeled material, the size of the elements and the time step size. The ODEs arising from volumetric deformation discretization are in general considered *stiff*, i.e., they impose a strict limit on the time step when using explicit time integration. Therefore, Debunne et al. analyze the Courant condition to devise the maximum admissible time step. In order to achieve constant animation rates, several intermediate steps must be computed which are then summed up.

Alternatively, the unconditionally stable *implicit time integration* scheme can be applied, which was introduced by Baraff and Witkin [BW98] into the computer graphics community. This leads to the update rules

$$\begin{pmatrix}\mathbf{u}_{(i+1)}\\ \dot{\mathbf{u}}_{(i+1)}\end{pmatrix} = \begin{pmatrix}\mathbf{u}_{(i)}\\ \dot{\mathbf{u}}_{(i)}\end{pmatrix} + \Delta t\begin{pmatrix}\dot{\mathbf{u}}_{(i+1)}\\ \mathbf{M}^{-1}\mathbf{f}_{(i+1)}\end{pmatrix}. \tag{2.89}$$

Implicit time integration is unconditionally stable, which is an important property as the equations of elasticity are considered stiff. The scheme is called implicit, because the values on the right-hand side are defined at the time step t_{i+1}, which are yet to be computed. In contrast, explicit schemes would require prohibitively small time steps to maintain stability. Eq. (2.89) can be rewritten as update rules for the change of displacement $\Delta\mathbf{u} = \mathbf{u}_{(i+1)} - \mathbf{u}_{(i)}$ and velocity $\Delta\dot{\mathbf{u}} = \dot{\mathbf{u}}_{(i+1)} - \dot{\mathbf{u}}_{(i)}$

$$\begin{pmatrix}\Delta\mathbf{u}\\ \Delta\dot{\mathbf{u}}\end{pmatrix} = \Delta t\begin{pmatrix}\dot{\mathbf{u}}_{(i+1)}\\ \mathbf{M}^{-1}\mathbf{f}_{(i+1)}\end{pmatrix}. \tag{2.90}$$

In order to estimate the forces at time step $i+1$, a Taylor expansion

$$\mathbf{f}_{(i+1)} \approx \mathbf{f}_{(i)} + \frac{\partial\mathbf{f}}{\partial\mathbf{u}}\Delta\mathbf{u} = \mathbf{f}_{(i)} - \mathbf{K}\Delta\mathbf{u} \tag{2.91}$$

can be set up and used as approximation. Inserting this into the lower equation of (2.90) and applying $\Delta\mathbf{u} = \Delta t\left(\dot{\mathbf{u}}_t + \Delta\dot{\mathbf{u}}\right)$ of the upper equation in the second step results in

$$\Delta\dot{\mathbf{u}} = \Delta t\mathbf{M}^{-1}\left(\mathbf{f}_{(i)} - \mathbf{K}\Delta\mathbf{u}\right) = \Delta t\mathbf{M}^{-1}\left[\mathbf{f}_{(i)} - \Delta t\mathbf{K}\left(\dot{\mathbf{u}}_{(i)} + \Delta\dot{\mathbf{u}}\right)\right]. \tag{2.92}$$

Rearranging the equation leads to

$$\left(\mathbf{M} + \Delta t^2\mathbf{K}\right)\Delta\dot{\mathbf{u}} = \Delta t\left(\mathbf{f}_{(i)} - \Delta t\mathbf{K}\dot{\mathbf{u}}_{(i)}\right), \tag{2.93}$$

which is a system of linear equations that must be solved to compute the change in velocity $\Delta\dot{\mathbf{u}}$. Afterwards the state of the system can be updated by computing

$$\dot{\mathbf{u}}_{(i+1)} = \dot{\mathbf{u}}_{(i)} + \Delta\dot{\mathbf{u}} \tag{2.94}$$

for the velocity and

$$\mathbf{u}_{(i+1)} = \mathbf{u}_{(i)} + \Delta t\dot{\mathbf{u}}_{(i+1)} \tag{2.95}$$

for the displacements.

2.1.4. Solving sparse linear systems

The implicit time integration scheme requires solving large linear systems of the form (see Eq. (2.93))

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \tag{2.96}$$

with $\mathbf{A} = \mathbf{M} + \Delta t^2\mathbf{K}$ and $\mathbf{b} = \Delta t\left(\mathbf{f}_{(i)} - \Delta t\mathbf{K}\dot{\mathbf{u}}_{(i)}\right)$. The matrix $\mathbf{A}$ is sparse and is of dimension $\mathbb{R}^{3m\times 3m}$. The solution of this system is the change in velocity $\mathbf{x} = \dot{\mathbf{u}}$ and is the most time consuming computation in a simulation step. Therefore, it is important to use efficient schemes for the solution.

In general, sparse linear solvers are either direct or iterative methods. *Direct sparse* solvers are summarized in Davis' book [Dav06]. This type of solver is only suitable for moderately large problems and is difficult to parallelize. Furthermore, they often involve factorization of the matrix with back substitution, only return the final solution and are not able to deliver intermediate, approximate results. Examples are Cholesky, LU and QR decompositions and implementations are available in libraries such as PARDISO [SG04] or Eigen [GJ*10]. For deformation simulation, a direct sparse solver was applied by Mezger et al. [MTPS08], who solve corotational elasticity with a constant frame rate guarantee. There, the runtime of the solver only depends on the size of the linear system and is not influenced by the elasticity properties of the body. However, as the linear system changes in every time step the expensive factorization must be recomputed. Hecht et al. [HLSO12] propose an adapted Cholesky factorization for the simulation of deformable models. To reduce the computational effort of the factorization, they introduce the so-called *warp-cancellation corotation* where the factorization is only updated locally, where for elements whose rotation matrices change significantly.

In contrast, *iterative* solvers improve an initial guess in every iteration towards the final result (see Barrett et al. [BBC*94] for a good overview). *Gauss-Seidel*, *Conjugate Gradient* (CG) and *multigrid* are examples for this class of solvers that are adopted in computer animation. As direct solvers cannot deliver intermediate solutions or be terminated earlier, iterative methods for solving linear systems are preferred in computer graphics. Iterative schemes compute successive values $\mathbf{x}^i$ towards the solution $\mathbf{x}$ and improve the residual

$$\mathbf{r}^i = \mathbf{b} - \mathbf{A}\mathbf{x}^i, \tag{2.97}$$

```
1  def pcg(A, b, x):
2      i = 0
3      r = b − Ax
4      d = P^{−1} r
5      δ_new = r · d
6      δ_0 = δ_new
7      while i < i_max and δ_new > ε^2 δ_0:
8          q = Ad
9          α = δ_new / (d · q)
10         x = x + αd
11         r = r − αq
12         s = P^{−1} r
13         δ_old = δ_new
14         δ_new = r · s
15         β = δ_new / δ_old
16         d = s + βd
17         i = i + 1
```

Listing 2.2.: A preconditioned conjugate gradient (PCG) algorithm to iteratively solve $\mathbf{Ax} = \mathbf{b}$. $i_{\max}$ is the upper bound of iterations and ε the relative accuracy threshold for the residual norm.

which measures the error in the matrix norm. The norm of the residual $\|\mathbf{r}^i\|_2$ converges to zero if the current approximation $\mathbf{x}^i$ approaches the solution $\mathbf{x}$. However, it is not possible to make assumptions on the closeness of $\mathbf{x}^i$ to $\mathbf{x}$ if the residual norm is significantly larger than zero.

A standard choice in computer graphics is the CG method [She94]. It can be applied when both matrices $\mathbf{M}$ and $\mathbf{K}$ are symmetric and positive definite, which is true for the FE discretization derived in Section 2.1.2. The basic CG algorithm with preconditioning (PCG) is given in Listing 2.2. It computes successive approximations $\mathbf{x}^i$ to solve Eq. (2.96) until a relative improvement of the residual norm $\frac{\|\mathbf{r}^i\|_2}{\|\mathbf{r}^0\|_2} < \varepsilon$ is reached or if the iterations exceed the maximum $i_{\max}$. Preconditioning in Listing 2.2 is the symbolic application of the matrix inverse $\mathbf{P}^{-1}$ of the preconditioner matrix $\mathbf{P}$. This process improves the condition number of the matrix and is equivalent to solving

$$\mathbf{P}^{-1}\mathbf{Ax} = \mathbf{P}^{-1}\mathbf{b}. \tag{2.98}$$

Depending on the choice of preconditioner the number of iterations to reach the stopping criterion can be reduced significantly. Typically, more efficient preconditioners require more effort for construction. A particularly simple preconditioner is diagonal preconditioning. There, the diagonal entries of the system matrix $\mathbf{P} = \mathrm{diag}(\mathbf{A})$ are used, where the application of $\mathbf{P}^{-1}$ is simply the multiplication with $\frac{1}{a_{ii}}$, the inverse of the diagonal elements of the system matrix.

Although many libraries efficiently implement the CG method, such as Eigen [GJ*10], specialized versions are required in some applications. One example is the modified PCG method that was introduced by Baraff and Witkin [BW98]. It allows for user-specified constraints on the degrees of freedom within the linear system, without the need to alter the matrix and the right-hand side. Therefore, a filtering strategy is applied such that a

user can impose and change constraints and boundary conditions in an interactive simulation session without the costly need to eliminate the corresponding rows and columns in the matrix data structure.

2.2. Related work

Interactivity in simulations of dynamic volumetric deformation is one of the central goals of this thesis. In order to achieve this goal, contributions associated with the following three pillars are developed:

- *more accurate discrete representations*
- *efficient methods for linear systems*
- *data structures and methods for massively parallel computing*

In this section, work related to the simulation of dynamic volumetric deformation according to these three pillars is reviewed. Furthermore, early works and the evolution of deformation simulation are summarized in Section 2.2.1. In Subsection 2.2.1.1, approaches that are based on a continuum mechanical modeling for the simulation of volumetric deformation are summarized. There, methods that solve the PDEs of elasticity using FE discretizations on tetrahedral meshes are reviewed. Subsection 2.2.1.2 summarizes alternative approaches also based on the PDEs of elasticity that employ other types of numerical discretization. Section 2.2.2 summarizes previous work that apply higher order FE discretizations to simulate volumetric deformation, which is relevant to the first pillar. In Section 2.2.3, methods are discussed that focus on the second pillar, i.e., the efficient solution of linear systems. Subsection 2.2.3.1 covers multigrid approaches, which are the most efficient solvers for linear systems, as they have linear time complexity. In Subsection 2.2.3.2, related work is reviewed that focuses on the setup of the linear systems. Section 2.2.4 summarizes the developments of massively parallel computations with GPUs in the context of sparse matrix data structures and simulation of volumetric deformation, which is associated with the third pillar.

2.2.1. Simulation of dynamic volumetric deformation in computer graphics

Simulation of volumetric deformation has a long history in computer graphics. Almost three decades of research have generated continuously improving models and methods to improve the realism and the speed of deformation simulations. A survey by Gibson and Mirtich [GM97] reviews early work on deformable modeling in computer graphics. There, geometric and physically based methods such as mass-spring systems and methods based on continuum mechanics are summarized. Later, a further survey was presented by Nealen et al. [NMK*06]. Furthermore, a good summary of volumetric deformation simulation can also be found in the course materials by Müller et al. [MSJT08] and Sifakis and Barbic [SB12], where FE modeling and model reduction approaches are reviewed.

This thesis focuses on continuum mechanical modeling of physical effects in order to achieve a high degree of realism. However, there are many other approaches that obtain visually realistic deformation besides those based on the PDEs of elasticity. The course by Müller et al. [MSJT08] also covers mass-spring systems and position-based dynamics. For the latter, the state-of-the-art report was presented by Bender et al. [BMO*12] that summarizes the recent advancements in this field.

Several books cover the broad subject of continuum mechanical modeling. The text book by Fung [Fun94] gives a general overview. The FEM has been developed more than fifty years ago and therefore the amount of literature covering it is vast. This method is rooted in the engineering domain, where many variants have been developed. This large amount of work is not reviewed here in detail. However, a comprehensive description of the method can be found in the textbooks by Zienckiewicz and Taylor [ZT00] and Bonet and Wood [BW08]. Furthermore, an important concept in this thesis is the representation of polynomials in BB-form as a special kind of FE model. These polynomials are ubiquitous in computer graphics, especially in computer aided design and geometric modeling. This representation is primarily used for Bézier curves and surfaces. Farin's book [Far02]

is a good reference for the modeling of curves and surfaces with polynomials in BB-form whereas the book by Lai and Schumaker [LS07] covers the volumetric case.

2.2.1.1. Continuum mechanical modeling of elasticity

Terzopoulus et al. [TPBF87] were among the first to simulate deformable models in computer graphics with a continuum mechanical approach. They use a *Lagrangian* approach based on the PDE of elasticity, i.e., the computational mesh moves as it is deformed. For elasticity modeling, the non-linear *Green* strain tensor formulation is used to measure the degree of deformation and the governing PDEs are solved with a finite difference (FD) scheme. While the so-called *geometric non-linearity* handles large deformations correctly, it is generally expensive to evaluate. In order to accelerate the necessary calculations aiming at interactive simulations, several authors linearize the Green strain tensor. The resulting *Cauchy strain tensor*, i.e., a *geometrically linear* model is a good approximation for small deformations, but rotational movements are not handled correctly and cause volumetric distortion. However, the advantage is that the relation between the position of the deformable body and the forces acting on it is linear. Bro-Nielsen [BN98] applied Cauchy strain with tetrahedral FEs and linear basis functions for efficient surgery simulation, where only little rotational movement occurs. Picinbono et al. [PDA00] analyzed the effect of linearization in more detail and propose to use *St. Venant-Kirchhoff elasticity* (StVK) instead. This special form of PDE utilizes geometric non-linearity with linear and isotropic material parameters.

The *corotational formulation* is an alternative modeling of deformation that has both advantages, a linear force-position relationship and rotational invariance. There, the forces are locally evaluated in a coordinate system that is close to the rest state and are then applied in the current transformed coordinate system, as described in Section 2.1.2.4. Müller et al. [MDM*02] introduced the *warped stiffness* method where the non-rotated reference frame is computed per vertex. The volumetric distortion arising from rotational movement is reduced but ghost forces occur that induce in implausible behavior. Later, Müller and Gross [MG04] evaluated the elastic forces in a per element reference frame eliminating the ghost forces, as the forces are now guaranteed to sum up to zero. At the same time this rotationally invariant formulation was also proposed by Hauth et al. [HS04], who introduced the *corotational strain tensor*. However, a disadvantage of the corotational method is the additional computational effort to rebuild the stiffness matrix in every time step. To determine the polar decomposition they propose an iterative scheme minimizing the distance of the deformation gradient and the rotation matrix w.r.t. the Frobenius norm instead. Irving et al. [ITF04] presented a method to cope with inverted tetrahedral elements that occur when large forces are present. For inverted configurations, the polar decomposition of the deformation gradient may result in reflection matrices, which prevent the elements from recovering from the inverted state and lead to implausible behavior. Therefore, Irving at al. use an SVD to diagonalize the deformation gradient. If the determinant of an intermediate matrix is negative, a column is negated to ensure that the final result is actually a rotation matrix. Schmedding and Teschner [ST08] analyzed the process of element inversion and recovery in more detail. They additionally take the direction of inversion into account to recover from the state faster and more realistically. Corotational elasticity is widely adopted by many authors to achieve fast simulations while handling large deformations especially rotations, correctly. The efficiency of the method was demonstrated by Parker and O'Brien [PO09], who successfully used corotational linear FEs to realistically simulate deformation and fracture in a computer game with strict time constraints for the computation.

Applying the FEM corresponds to a spatial discretization rendering the PDE into a set of ODEs that still contain derivatives w.r.t. time. The process of eliminating the time derivatives and to generate rules for updating the state, i.e., the velocities and positions of the system, is called *time integration* and has been described in Section 2.1.3. The implicit integration scheme was introduced into the computer graphics community by Baraff and Witkin [BW98]. The discretization results in a linear or non-linear system of equations for the velocity

update depending on whether the ODE is linear or non-linear. For non-linear ODEs an iterative *Newton-Raphson* method can be applied, where every iteration requires the tangent matrix and a solution of a linear system. Hauth et al. [HES03] analyzed various time stepping schemes for computer animation in detail. They also investigate higher-order methods such as *Runge-Kutta* or their implicit counter parts, so-called *backwards differentiation formulas*, and analyze them regarding their accuracy and stability. The majority of researchers utilize this scheme for linear ODEs, mainly because of its unconditional stability [MG04, ST08, GW08]. However, in contrast to explicit time integration a large sparse linear system has to be solved.

Tetrahedral linear FEs with implicit time integration are widely adopted in computer graphics applications. The methods summarized here use mathematically simplified models to simulate volumetric deformation more quickly compared to engineering approaches. However, the linear basis functions imply a low approximation quality, which requires many elements to be able to represent complex deformations.

In its original form, corotational elasticity implicitly approximates the force derivatives, as it does not take the dependency of the rotation matrices w.r.t. the positions of the mesh into account. This approximation also contributes to the jitter effects observed by Georgii et al. [GW08]. *Exact corotational elasticity* provides a remedy to this, as this dependency is explicitly considered there. Chao et al. [CPSS10] propose a simple geometric model for elasticity, where the rotational derivatives are taken into account. Furthermore, they demonstrate that using this approach with a fully variational time integrator a long-term energy and momentum conservation can be achieved. However, the force gradients are then not linear anymore leading to non-linear systems of equations that are more expensive to solve. Barbic and Zhao [BZ11] build upon exact corotational elasticity and simulate open-loop deformable structures and explicitly compute the derivatives of the polar decompositions for the force gradients. McAdams et al. [MZS*11] also adopted the exact corotational method using SVD on regular lattices with a FD discretization. Additionally, they propose an indefiniteness correction for the stiffness matrix. Stomakhin et al. [SHST12] also observed that the corotational method results in jitter with large stretched configurations. They noticed that unstable energy minima cause non-physical inversion and oscillation and develop a smooth extension for hyperelastic energies to handle inverted configurations more robustly. In their practice and experience paper, Sin et al. [SSB13] present the Vega library, which provides several implementations, detailed analyses and guidelines for implementing different methods for simulating volumetric deformation.

In order to reduce the computational effort caused by the implicit time integration, Fierz et al. [FSH11] proposed *Implicit-Explicit time integration*. This scheme combines expensive implicit and cheap explicit integration locally. Based on the geometry of the tetrahedral elements and the time step size, nodes in the mesh are identified that require implicit treatment, while explicit integration is applied for the rest of the mesh. Michels at al. [MSW14] introduced an exponential integrator for systems of ODEs that exhibits long-term energy conservation. The stiff part of the ODE is solved with a closed form solution based on matrix exponentiation and the evaluation of the resulting trigonometric functions. This class of integrator allows for significantly larger time steps without numerical damping and energy is therefore conserved.

The aforementioned methods represent extensions and improvements to the standard corotational method and implicit time integration. Combining these with the approaches developed in this thesis may be an interesting option for future work.

2.2.1.2. Alternative discretizations of elasticity

Many computer animation papers based on continuum mechanics use corotational FEs on tetrahedral meshes for computing deformation. This may be due to the fact that the corotational FEM is an efficient and stable method, and tetrahedral meshes are able to conform with triangular surface meshes. Many other approaches have been proposed to solve the PDEs of elasticity. Early work by James and Pai [JP99] applied a *boundary element method*

to solve the PDEs with a quasi-static approach. In this method, no volumetric representation of the solid body is required and the corresponding equations are directly solved on a surface mesh discretization. As a drawback, the resulting system of linear equations is dense, which is challenging to solve efficiently when a large number of degrees of freedom is present. Grinspun et al. [GKS02] introduced a framework for adaptive simulation. They utilize FEs with hierarchical basis functions that can be refined without altering the computational mesh. Teran et al. [TBHF03] proposed to use a finite volume method on tetrahedral meshes to simulate nearly incompressible muscle tissue with non-linear material models. For constant strain elements this method is identical to an FE discretization with linear basis functions. Wicke et al. [WBG07] developed an FEM for convex polyhedra further broadening the options for the topology of the input mesh. As standard FE basis functions are not suitable for convex polyhedra, 3D mean value coordinates are chosen to serve as basis functions for representing the field of displacements. This approach was extended by Martin et al. [MKB*08] so that non-convex polyhedra can be handled as well. Harmonic basis functions are chosen and deformation is simulated using corotational elasticity. Kaufmann et al. [KMBG08] introduced the *Discontinuous Galerkin method* (DGM) into the computer graphics community to simulate volumetric deformation. The advantage is that the computational mesh can be composed of different types of elements, such as tetrahedra, convex and non-convex polyhedra. This is especially beneficial when considering local adaptivity or modeling fracture and cutting, as there are no restrictions on the types of elements of the computational mesh. However, the number of degrees of freedom rises significantly, as the resulting field is not required to be continuous across element boundaries for the DGM and unknowns at element boundaries are duplicated. Later, Kaufmann et al. [KMB*09] presented the *extended finite element method*, which is applied to the modeling of cuts and cracks in shell-like models by locally refining basis functions without altering the computational mesh. A specialized embedding technique for efficient animation of surface meshes was presented by Nesme et al. [NKJF09]. There, the simulation is conducted with a coarse hexahedral FE discretization but contact and interaction is realized via highly detailed surface meshes. In addition, the topology of the surface mesh is taken into account by inserting topological branches into the coarse computational mesh. McAdams et al. [MZS*11] used regular lattices as a special case of hexahedral elements for discretization to achieve nearly interactive and stable deformation simulation using a FD method. A quasi-meshless approach has been presented by Faure et al. [FGBP11] that uses Voronoi kernel functions to represent local frames for simulating volumetric deformation.

In order to significantly speed up the computations for real-time scenarios, Barbic and James [BJ05] developed a model reduction method with subspace integration and geometric non-linearity. There, a reduced basis representing the major deformation modes is precomputed and is used to determine deformation at runtime. This approach is extended by Zhao and Barbic [ZB13], who use arbitrary triangular meshes of plants as user input and automatically generate the corresponding reduced order simulation models. Furthermore, they support fracture of open loop geometry that occurs when forces become too large. Subspace methods model deformation with a small number of degrees of freedom. When collisions are present, this restricted representation may not be able to capture the required details.

An alternative to Lagrangian modeling is the so-called *Eulerian* approach, which recently gained popularity in the computer graphics community. Instead of moving and deforming the computational mesh, a fixed, non-moving spatial discretization is used to solve the PDEs of elasticity. This approach is similar to many methods in computational fluid dynamics that also use fixed computational meshes. Levin et al. [LLJ*11] introduced the Eulerian approach for deformation simulation in computer graphics and highlight its advantage in implicit collision detection between multiple objects. There, a binary indicator function is defined on the mesh that marks cells if they contain the deformable solid. The concept was extended by Fan et al. [FLLP13] by mixing in Lagrangian aspects, bridging the gap between simulation of Eulerian and Lagrangian approaches. In addition, they transform the computational mesh by translating and rotating it, but simulate deformation within the box in an Eulerian fashion. This way, global motion can be represented, which was restricted to the size of the

Eulerian mesh in previous work. The method is extended in the work of Fan et al. [FLP14] to model volumetric musculoskeletal systems, where tissues, muscles, collision and contact between them are computed accurately.

The methods summarized in this section are alternative formulations or further extensions to simulate visually plausible dynamic volumetric deformation. They might also benefit from the efficient methods developed in this thesis.

2.2.2. Higher-order FEs for deformation simulation

All of the previously mentioned methods have in common that the local spatial resolution determines the richness of detail of the deformation. For FEs, this is directly related to the number of basis functions that coincides with the number of degrees of freedom of the system. More degrees of freedom allow for a more accurate representation of deformation. In this section, the approaches reviewed can be associated with the pillar *more accurate discrete representations* as an alternative to increase the number of elements in the simulation for higher accuracy. As an option, the modeling power of the individual element can also be increased, e.g., by replacing the commonly used linear basis functions with *higher-order* bases. In general, simulation with higher-order FEs is wide-spread in engineering applications (see e.g. [ZT00]), where linear shape functions do not provide sufficient accuracy. In the context of computer graphics, Mezger et al. [MS06] introduced quadratic basis functions for animation and used the Green strain tensor to measure deformation. They report that the number of elements can be significantly reduced compared to linear basis functions while maintaining the accuracy of the simulation. Furthermore, for nearly incompressible materials shear locking is avoided, a problem which is also known in the engineering community with the advice to use at least quadratic basis functions for tetrahedral elements [ZT00]. In [MTPS08], Mezger et al. improve the speed for higher-order FE simulation by using the corotational elasticity method instead of a geometrically non-linear model. The numerical integration of the strain energy has to be evaluated at carefully selected points in order to maintain the quadratic convergence of higher-order elements. Roth et al. [RGTC98] discussed the modeling of soft tissue in the context of facial surgery simulation for static, i.e., time-independent cases. They used higher-order polynomial basis functions in BB-form to model the displacement field. However, their focus was on high accuracy for a static simulation instead of interactive simulation. Recently, Bargteil and Cohen [BC14] presented a framework for adaptive FEs with linear and quadratic BB-form polynomials. They build upon the work developed in this thesis and cited the corresponding publications [WKS*11] and [WBS*13]. Additionally, they apply p-adaptivity, i.e., they start with linear basis functions and raise the degree per tetrahedron when higher accuracy is needed.

Early approaches have applied standard quadratic FEs to simulate dynamic volumetric deformation in order to speed up the computations. Furthermore, BB-form FEs were utilized but only in static deformation approaches. BB-form methods for the simulation of dynamic volumetric deformation and efficient assembly methods for linear systems are still missing to significantly accelerate the overall computation.

2.2.3. Solving linear systems

At the core of many methods, a large sparse linear systems has to be solved to update the state of the system. In many cases, this is the most time intensive operation when computing the new state of the system at a new time step. Therefore, the methods reviewed in this section belong to the pillar *efficient methods for linear systems*. Several solvers that can be used to compute a solution for the corresponding linear systems are described in Section 2.1.4. Many works in the area of deformation simulation utilize the CG method, as it is easy to implement and provides an efficient way of solving large, sparse linear systems [NMP*05, TSIF05, SSB13]. Alternatively, Mezger et al. [MTPS08] utilize a sparse direct method for corotational elasticity to guarantee a constant frame

rate. There, the runtime of the solver only depends on the size of the linear system and is not influenced by the elasticity properties of the body. However, as the linear system changes in every time step the expensive factorization must be recomputed.

2.2.3.1. Multigrid approaches

Multigrid methods are generally the most efficient solvers for linear systems, as their runtime is linear w.r.t. the number of degrees of freedom. They have been the subject of extensive research and standard textbooks like [TS01] and [BHM00] that provide a good overview of the basic method and its theory. For corotational elasticity with tetrahedral meshes, Georgii et al. [GW06] proposed a *geometric multigrid* algorithm based on linear FEs using nested and non-nested mesh hierarchies. In general, this geometric multigrid concept cannot be easily adapted for higher-order FEs. In their work, the computational bottleneck is the matrix update, where sparse matrix-matrix products (SpMM) are required on every level to update the multigrid hierarchy. Later, they specifically developed an optimized SpMM implementation to address this bottleneck and report a speedup of one order of magnitude [GW08]. However, the matrix update is still as expensive as applying the multigrid algorithm itself. Geometric multigrid methods are especially well suited for discretizations on regular grids, as the inter-grid transfer operators are straightforward to derive. In the context of deformation simulation, Zhu et al. [ZSTB10] propose a multigrid framework based on FDs. Dick et al. [DGW11b] use hexahedral FE discretization on a regular grid and solve the linear systems using a multigrid method. In [DGW11a], they extend this approach to simulate cuts. p-multigrid methods on the other hand are specialized methods that act on hierarchies of polynomial representations. For two-dimensional analysis of elliptic boundary value problems such as the Poisson equation, Shu et al. [SSX06] introduced a p-multigrid method for FEs using a higher-order Lagrangian basis. By utilizing the restriction and prolongation operators, they reconstruct the linear discretization from the higher-order discretization. However, they do not directly discretize the PDE with varying polynomial degree, which then requires expensive SpMM products.

Up to now, no polynomial multigrid method exists for three-dimensional tetrahedral mesh discretizations and for volumetric deformation. Furthermore, hierarchies of polynomial BB-form FEs for a p-multigrid approach have not yet been developed.

2.2.3.2. Assembling linear systems

A further relevant, time-intensive building block is the construction of the sparse matrix data structure, the so-called *matrix assembly*, especially if the matrix changes in every time step. This is the case when applying corotational or non-linear elasticity. Rebuilding the matrix is also necessary if topological changes or adaptivity require changing the discretization mesh. Then, it is required to recompute the element matrix entries and to assemble the global matrix data structure. In order to avoid the expensive matrix assembly, Nesme et al. [NPF05] proposed to emulate SpMV for a CG method by iterating over the individual element contributions. This is a reasonable approach when the time required to generate the matrices is higher than the performance loss that is induced by every SpMV operation for the CG method. To accelerate the matrix construction, Georgii et al. [GW08] directly stored references from element matrices to global matrices. Similarly, Kikuuwe et al. [KTY09] introduced a specialized edge[1] data structure to speed up the construction for non-linear modeling with StVK materials. They associate parts of the element stiffness entries with the edges and introduce the concept of tetrahedron sharing edge pairs to efficiently assemble the global (tangent) stiffness matrix. Peña Serna et al. [PnSdSSM09] also studied the properties of matrices arising from a tetrahedral mesh discretization for deformation simulation with

[1] The edges connecting vertices in the tetrahedral mesh determine the non-zero entries in the sparse matrix of linear FE discretizations. By iterating over all edges of a mesh, an efficient matrix assembly can be realized.

linear basis functions and geometric linear modeling. They analyzed the relationship between topological elements and the resulting structure of the matrix and directly use the mesh as a proxy for the sparse matrix data structure. This method allows for efficient tetrahedral mesh editing in the context of simulation, especially when the topology of the mesh is changed.

Previous methods are specialized for linear FEs and cannot be applied when utilizing higher-order FE approaches. In order to achieve an efficient construction of the linear system for higher-order discretization, novel methods are required.

2.2.4. GPU-based methods

Linear algebra operations are generally suitable for parallelization and therefore offer a high potential for acceleration. Therefore, this section reviews methods associated with the pillar *data structures and methods for massively parallel computing*. *Massively parallel computing* is the concept of executing algorithms on architectures with a large number of processors. In this thesis the focus is on massively parallel algorithms executed on GPUs. If the method is amenable to parallelization, the runtime can be reduced significantly by developing specialized data structures and algorithms. Early works by Bolz et al. [BFGS03] as well as Krüger and Westermann [KW03] used graphics hardware to speed up computations for solving systems of linear equations. Both used the graphics pipeline with programmable shaders and textures to represent operations and data, respectively. With the introduction of NVIDIA's *Compute Unified Device Architecture* (CUDA) [NVI15] to abstract from the underlying graphics hardware, general-purpose computing on graphics processing units (GPGPU) was increasingly interesting. Among a lot of work for dense matrices, a few methods have been presented to handle sparse matrices. In particular, the library cuSPARSE [NVI11] has been released providing GPU-accelerated BLAS (basic linear algebra subprograms) operations and SpMV. The work of Bell et al. [BG08, BG09] presents different optimized data structures for sparse matrices. There, the performance of well-known formats such as *diagonal (DIA)*, *ELLPACK (ELL)*, *compressed sparse row (CSR)* and the *coordinate (COO)* on GPU hardware is analyzed. Furthermore, the *hybrid format (HYP)* is introduced which combines the COO and the ELL format for sparse matrices with a varying number of non-zero entries per row. The open source library CUSP [BG10] is based on this work and clearly exceeds the performance of the cuSPARSE library for matrices with highly varying row lengths. An alternative approach has been presented by Buatois et al. [BCL09] where the clustering of non-zero entries in the sparse matrix is identified to permit register blocking and optimized memory fetches. Baskaran et al. [BB09a] focus on coalesced loading of sparse matrix and data reuse and report a further acceleration of SpMV operations with matrices in CSR-format. Vazquez et al. [VOFG10] proposed an optimization of the ELL-format for GPUs called *ELLPACK-R* by additionally storing the row length and transposing the data to achieve coalesced memory transactions. This approach is generalized by the *Sliced ELLPACK* format by Monakov et al. [MLA10] grouping rows together. However, for this data structure the SpMV operation needs additional kernel calls for a subsequent reduction. Oberhuber et al. [OSV11] introduced the *row-grouped CSR* format, but without clustering non-zero elements.

Some of these sparse matrix data structures are well-known representations for scientific computing. The aforementioned works evaluated how they behave on GPUs. In addition, optimized versions adapted to the special requirements of GPUs have been developed. However, no data structure exists yet that allows for a high degree of coherent memory access, while considering the special structure of matrices from deformation simulation.

Although methods that simulate deformation would greatly benefit from using only GPU-accelerated linear algebra, it is additionally worth coming up with a parallelization for the whole method. This is due to the fact that other building blocks may also benefit from parallelization and data transfers between CPU and GPU memory

would imply a performance hit. Early work simulating deformation on the GPU using programmable shaders and textures for medical training scenarios was presented Wu and Heng [WH04]. They utilize geometrically linear FEs with a quasi-static approach and solve the linear system with a CG method. Liu et al. [LJWD08] follow a similar approach and use CUDA to implement geometrically linear FEs and solve the linear systems with a CG algorithm. A GPU implementation for geometrically non-linear FEs with explicit time integration was presented by Taylor et al. [TCO08]. Courtecuisse and Allard [CA09] presented a parallel Gauss-Seidel algorithm to solve a dense linear complementarity problem on multi-core CPUs or GPUs. This is used for efficient contact handling between deformable bodies. A GPU-based simulation of elastic bodies using corotational elasticity was presented by Dick et al. [DGW11b]. They use a lattice to discretize the simulated geometry and develop a multigrid algorithm for solving the linear system. Later, Allard et al. [ACF11] presented a GPU-based approach to simulate deformable models with tetrahedral elements. They propose to emulate the SpMV operation by splitting the computations down to the individual element contributions. This is a reasonable approach when the time required to generate the matrices is higher than the performance loss that is induced by every matrix-vector multiplication. Furthermore, they also proposed an approach for merging the GPU kernel calls in a CG solver. This method is similar to the one proposed in Section 2.5 and has been developed in parallel.

However, the combination of massively parallel computations with higher-order FEs has not yet been investigated. Furthermore, none of the methods employ a highly optimized sparse matrix data structure for simulations of dynamic volumetric deformation. An efficient matrix update on GPUs for the corotational method has not yet been investigated.

2.2.5. Summary

A common approach for simulating dynamic volumetric deformation is to apply a linear FE discretization on tetrahedral elements with implicit time integration. This serves as a reference for the methods developed in this chapter. The most time consuming parts in such a scenario are the setup and the solution of the linear system as shown in Section 2.3. An interesting option to reduce the size of the linear system is to spend significantly fewer elements but increasing the order of the discretization at the same time. The application of quadratic polynomials to the FE discretization has been proposed by a few authors for improving accuracy and speed in simulations. In parallel, several authors developed a multigrid algorithm for linear FEs to efficiently solve the system of equations. Other works accelerate the setup of the linear system by developing special methods for linear FEs. The option to make use of massively parallel computing for simulation applications has also attracted many authors. Several data structures and methods for solving the linear systems in deformation simulation scenarios have been presented, which significantly accelerate the computations.

However, open questions remain, especially whether computation time and accuracy can be further improved. The combination of BB-form higher-order discretizations and dynamic volumetric simulation has not yet been investigated. Furthermore, efficient methods for constructing the linear systems based on higher-order discretizations have not yet been designed. The multigrid approaches in previous work are only applicable to linear FEs. Methods that explicitly make use of the polynomial hierarchies on three-dimensional tetrahedral mesh discretizations do not exist. Furthermore, the exploitation of massively parallel computing for dynamic volumetric deformation can be improved. Optimized data structures specialized for deformation simulation are missing. In addition, quadratic BB-form FEs have not yet been simulated with a massively parallel approach.

The issues summarized here require further investigation and are addressed in the following sections.

2.3. The quadratic BB-form method

In this section, a novel physically based interactive simulation technique for volumetric deformation with higher-order bases for FEs is presented. Based on corotational elasticity, the method models geometry as well as displacements using quadratic basis functions in BB-form on a tetrahedral FE mesh. The BB-formulation yields significant advantages compared to approaches using the monomial form of polynomials. The implementation is simplified, as spatial derivatives and integrals of the displacement field are obtained analytically, avoiding to numerically evaluate the elements' stiffness matrices. Furthermore, the BB-form allows for an approximation of the rotation as the integrals do not require to be evaluated at specified integration points. Additionally, a novel traversal accounting for adjacency in tetrahedral meshes is introduced in order to accelerate the reconstruction of the global matrices. It is shown that the proposed method can compensate the additional effort introduced by the corotational formulation to a large extent. The approach is validated on several models and demonstrates new levels of accuracy and performance in comparison to current state of the art. This section is based on the publication [WKS*11], which itself builds up on the diploma thesis [Web08]. Large parts of the publication's text are copied verbatim with minor additions and corrections. Furthermore, pictures and diagrams are reused as well.

2.3.1. Introduction

A common issue in interactive deformation simulation using FEMs is the trade-off between speed and accuracy of the solution. While adding more elements improves the accuracy of the simulation, the size of the resulting system often leads to prohibitively slow simulations. Instead of increasing the number of elements in the simulation, the modeling power of the individual element can also be increased, e.g., by replacing the commonly used linear basis functions with higher-order bases. Mezger et al. [MS06, MTPS08] follow this approach and use quadratic polynomials in a monomial basis to model deformable bodies. They achieve interactive rates by reducing the number of elements and attaching a detailed surface mesh. However, implementation is more complex compared to linear elements. Coefficients of basis functions must be determined by solving linear systems beforehand and numerical integration must be applied.

Here, a new method is proposed to model the deformation field using quadratic interpolation functions in BB-form. It is demonstrated that this alternative representation leads to a concise description and yields a simpler implementation. This has only been done once before by Roth et al. [RGTC98, Rot02] in the context of static simulation of tissue with a focus on high accuracy and C^1 continuity of the resulting surface. While it does not

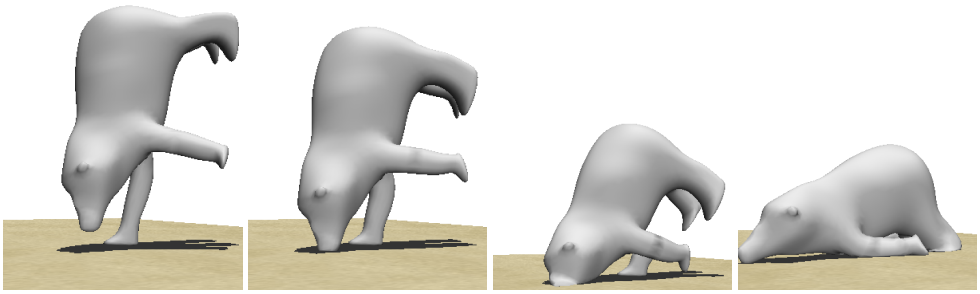


Figure 2.15.: Interactive deformation simulation of the Bear model with 1022 quadratic tetrahedra.

enforce C^1 continuity, the approach reveals how the BB-form yields a more elegant approach than existing FE methods based on quadratic elements.

Current state-of-the-art approaches for interactive deformation simulation are based on a corotational approach in order to handle large deformations. Analyzing performance evaluations of work in that field (e.g., [MTPS08, GW08, PO09]) creates the impression that solving the system of linear equations is no longer the crucial bottleneck in the simulation loop. Instead, a considerable amount of simulation time is spent to extract rotations and reassemble the global stiffness matrix. This part of the simulation step is analyzed and the usage of quadratic FEs is proposed to reduce the number of rotation extractions. In contrast to Mezger et al. [MTPS08], rotations are approximated with one matrix per element instead of four in the proposed method. Furthermore, an optimized matrix assembly for quadratic FEs is presented accelerating the construction of the global stiffness matrix. It is shown that the use of quadratic elements (with a comparable number of degrees of freedom) in the corotational setting is faster and shows superior accuracy compared to linear elements. The contributions are as follows:

- A Bernstein-Bézier-based approach for static simulation [RGTC98, Rot02] is combined with the corotational formulation of Mezger et al. [MTPS08] to handle large deformations.
- It is shown that modeling with trivariate polynomials in BB-form can facilitate calculations and is an elegant alternative representation for deformation simulation. The mathematical formulation simplifies the implementation of the simulation algorithm compared to other formulations in monomial basis.
- A framework for quadratic FEs is presented that is able to handle element inversion.
- An efficient, mesh oriented traversal for quadratic basis functions is introduced accelerating the construction of the global stiffness matrix.
- It is demonstrated that quadratic elements are superior to linear ones w.r.t. speed and accuracy.
- It is shown that simulations of models with thousands of degrees of freedom can be interactive using quadratic bases to model object geometry and deformation.

2.3.2. Trivariate Bernstein-Bézier-form

The basis functions of the FE solver are given in BB-form (see [Far02, LS07] for more details). The major advantages are:

- Polynomials are defined w.r.t. a tetrahedral element making them coordinate frame independent.
- There is no need to precompute polynomial coefficients. Stiffness and mass matrix entries are independent of the basis functions.
- All operations on polynomials in BB-form, including integral and differential operations, can be evaluated analytically and efficiently.

A piecewise polynomial representation of the field of unknowns is constructed on a given tetrahedral mesh $\triangle := (\mathcal{T}, \mathcal{F}, \mathcal{E}, \mathcal{V})$ with a set of tetrahedra $\mathcal{T}$, faces $\mathcal{F}$, edges $\mathcal{E}$ and vertices $\mathcal{V}$. This discretization is generated from surface meshes with the CGAL library [CGA]. The union of all elements forms a non-uniform partition of the simulation domain $\Omega = \bigcup_{\mathbb{T}\in\mathcal{T}} \mathbb{T}$. Two tetrahedra of the mesh are either disjoint or share a face, an edge or a vertex. The fields on the simulation domain Ω are defined by one polynomial per element. A degree-p polynomial

$$f|_{\mathbb{T}} = \sum_{i+j+k+l=p} b_{ijkl} B_{ijkl}(\mathbf{x}), \quad i,j,k,l \geq 0, \tag{2.99}$$

is defined w.r.t. a tetrahedron $\mathbb{T} = [\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3]$.

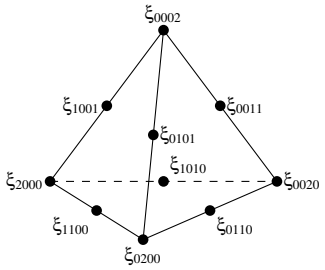


Figure 2.16.: Domain points for a quadratic Bézier tetrahedron. Points at the vertices have exactly one non-zero index.

Here, the unknowns b_{ijkl} and basis functions B_{ijkl} are associated with the domain points or nodes

$$\xi^{\mathbb{T}}_{ijkl} = \frac{i\mathbf{p}_0 + j\mathbf{p}_1 + k\mathbf{p}_2 + l\mathbf{p}_3}{p}, \quad i+j+k+l = p. \tag{2.100}$$

This is a representation of the parametric domain, where a coefficient b_{ijkl} may be a scalar, a vector or a tensor. In the case of quadratic elements, ten coefficients b_{ijkl} are associated with the domain points (see Fig. 2.16). Then, similar to Bézier curves, the values associated with the vertices of $\mathbb{T}$ (b_{2000}, b_{0200}, b_{0020} and b_{0002}) are given by interpolation, the six remaining edge coefficients are defined implicitly by derivatives and positions at the vertices. The total number of degrees of freedom for the tetrahedral mesh is given by the union of all tetrahedral nodes $\mathcal{D}_\triangle = \bigcup_{\mathbb{T}\in\mathcal{T}} \bigcup_{i+j+k+l=p} \xi^{\mathbb{T}}_{ijkl}$. This set can be indexed, i.e., global indices α can be mapped to nodes $d_\alpha \in \mathcal{D}_\triangle$, where $\alpha = 0,\dots,N-1$, $N = |\mathcal{D}_\triangle|$. The $\binom{p+3}{3}$ blending or shape functions in Eq. (2.99) are the *Bernstein polynomials*

$$B_{ijkl} = \frac{p!}{i!j!k!l!}\lambda_0^i\lambda_1^j\lambda_2^k\lambda_3^l, \quad i+j+k+l = p, \tag{2.101}$$

and form a basis of the space of degree-p polynomials $\mathcal{P}_p$ [LS07]. Here, the *barycentric coordinates* $\lambda_\nu \in \mathcal{P}_1$ are the linear polynomials determined by $\lambda_\nu(\mathbf{v}_\mu) = \delta_{\nu,\mu}$. They represent the volume fractions that arise from splitting an element at $\mathbf{x}$ into four tetrahedra: $\lambda_i = \frac{V_i}{V_\mathbb{T}}$. The volume of an element $\mathbb{T}$ is $V_\mathbb{T} = \frac{1}{6}\det\begin{pmatrix}\mathbf{p}_0 & \mathbf{p}_1 & \mathbf{p}_2 & \mathbf{p}_3\\ 1 & 1 & 1 & 1\end{pmatrix}$. V_i can be computed by replacing the i-th vertex with $\mathbf{x}$. In the case of a monomial representation, the basis function are

$$N_{ijkl}(\mathbf{x}) = \sum_{i+j+k+l=p} \alpha_{ijkl}x^iy^jz^k, \quad i,j,k,l \geq 0, \tag{2.102}$$

where the coefficients α_{ijkl} must be precomputed. They can be determined by solving a linear system of equations for each element [MS06, MTPS08]. This precomputation is not necessary when adopting the BB-form.

In the following, it is shown how the BB-form allows for analytic computation of integrals independent of the position and orientation of the underlying tetrahedron. Only the element's volume and differential changes of the barycentric coordinates are incorporated in the computations.

Directional Derivatives The partial derivatives $\frac{\partial B_{ijkl}}{\partial x_i}$ can be expressed as a sum of Bernstein polynomials of degree $q-1$ by applying the chain rule:

$$\frac{\partial B_{ijkl}}{\partial x_a} = \sum_{c=0}^{3} \frac{\partial B_{ijkl}}{\partial \lambda_c}\frac{\partial \lambda_c}{\partial x_a} = p\left[B_{i-1jkl}\frac{\partial \lambda_0}{\partial x_a} + B_{ij-1kl}\frac{\partial \lambda_1}{\partial x_a} + B_{ijk-1l}\frac{\partial \lambda_2}{\partial x_a} + B_{ijkl-1}\frac{\partial \lambda_3}{\partial x_a}\right], \tag{2.103}$$

where Bernstein polynomials with negative indices are set to 0. Note that for quadratic polynomials at least two of the terms in Eq. (2.103) are equal to zero. As the barycentric coordinates are linear polynomials, their partial derivatives are constant during the simulation and can be precomputed.

Multiplication The multiplication of two Bernstein polynomials of degree p on $\mathbb{T}$ is computed by

$$B^p_{ijkl} \cdot B^p_{mnoq} = G(i,j,k,l,m,n,o,q) B^{2p}_{(i+m)(j+n)(k+o)(l+q)}$$

with $i+j+k+l = m+n+o+q = p$. Using multi-indices $(I,J) = ((i,j,k,l),(m,n,o,q))$, the function G consists of binomials: $G(I,J) = \frac{\binom{i+m}{i}\binom{j+n}{j}\binom{k+o}{k}\binom{l+q}{l}}{\binom{2p}{p}}$.

Integration The integral of a Bernstein polynomial w.r.t. $\mathbb{T}$ is solely dependent on the degree of the polynomial and the volume $V_{\mathbb{T}}$:

$$\int_{\mathbb{T}} B_{ijkl} \,\mathrm{d}V = \frac{V_{\mathbb{T}}}{\binom{p+3}{3}}. \tag{2.104}$$

Therefore, the integral of a polynomial in BB-form can be easily evaluated by summation:

$$\int_{\mathbb{T}} f|_{\mathbb{T}} \,\mathrm{d}V = \frac{V_{\mathbb{T}}}{\binom{p+3}{3}} \sum_{i+j+k+l=p} \mathbf{b}_{ijkl}. \tag{2.105}$$

2.3.3. The quadratic BB-form method

2.3.3.1. Computing Matrix Entries

In Section 2.1.2.2, the general procedure to determine entries for the element matrices is described. Now, the computation of these entries when utilizing polynomials of degree two in BB-form is presented. The element matrices for a tetrahedron $\mathbb{T}$ can be divided into contributions for all control points with multi-indices (I,J). Thus, a 3×3 entry is computed by

$$\left[\mathbf{M}_{\mathbb{T}_{(I,J)}}\right]_{ab} = \int_{\mathbb{T}} \rho(B_{ijkl} B_{mnoq}) \delta_{ab} \,\mathrm{d}V, \tag{2.106}$$

or

$$\left[\mathbf{K}_{\mathbb{T}_{(I,J)}}\right]_{ab} = \int_{\mathbb{T}} \lambda_L \frac{\partial B_{ijkl}}{\partial x_a}\frac{\partial B_{mnoq}}{\partial x_b} + \mu_L \frac{\partial B_{ijkl}}{\partial x_b}\frac{\partial B_{mnoq}}{\partial x_a} + \mu_L \Big(\sum_{c=0}^{2} \frac{\partial B_{ijkl}}{\partial x_c}\frac{\partial B_{mnoq}}{\partial x_c}\Big)\delta_{ab} \,\mathrm{d}V, \tag{2.107}$$

respectively. The integrals of products of degree two Bernstein polynomials have to be evaluated for the mass matrix. As shown in Section 2.3.2, this results in a polynomial of degree four. With B^p indicating a degree p polynomial and binomial coefficients $G(I,J)$ arising from the multiplication, the integration results in

$$\int_{\mathbb{T}} B^2_{ijkl} B^2_{mnoq} \,\mathrm{d}V = G(I,J) \int_{\mathbb{T}} B^4 \,\mathrm{d}V = G(I,J) \frac{V_{\mathbb{T}}}{\binom{7}{3}}. \tag{2.108}$$

The contributions for $\mathbf{K}$ in Eq. (2.107) consist of several linear combinations of terms such as $\int_{\mathbb{T}} \frac{\partial B_{ijkl}}{\partial x_a}\frac{\partial B_{mnoq}}{\partial x_b} \,\mathrm{d}V$. The differentiation of a Bernstein polynomial w.r.t. the coordinate axis a is given in Eq. (2.103). Note that

polynomials with negative indices vanish, so at most two terms in the sum are non-zero. Rearranging leads to

$$\begin{aligned}\int_{\mathbb{T}} \frac{\partial B_{ijkl}}{\partial x_a}\frac{\partial B_{mnoq}}{\partial x_b}\,\mathrm{d}V &= \int_{\mathbb{T}} \left(\sum_{c=0}^{3}\frac{\partial B_{ijkl}}{\partial \lambda_c}\frac{\partial \lambda_c}{\partial x_a}\right)\left(\sum_{d=0}^{3}\frac{\partial B_{mnoq}}{\partial \lambda_d}\frac{\partial \lambda_d}{\partial x_b}\right)\mathrm{d}V \\ &= \sum_{c,d=0}^{3}\frac{\partial \lambda_c}{\partial x_a}\frac{\partial \lambda_d}{\partial x_b}\int_{\mathbb{T}}\frac{\partial B_{ijkl}}{\partial \lambda_c}\frac{\partial B_{mnoq}}{\partial \lambda_d}\,\mathrm{d}V.\end{aligned} \tag{2.109}$$

As the differentiation of $\frac{\partial B^2}{\partial \lambda_c} = 2B^1$ results in a linear polynomial, the integral is

$$\int_{\mathbb{T}}\frac{\partial B_{ijkl}}{\partial \lambda_c}\frac{\partial B_{mnoq}}{\partial \lambda_d}\,\mathrm{d}V = 4G(I,J)\int_{\mathbb{T}} B^2\,\mathrm{d}V = \frac{2}{5}G(I,J)V_{\mathbb{T}}.$$

Here, no numerical integration or evaluation of any basis function is needed. Only binomial coefficients, the element's volume and directional derivatives of the barycentric coordinates affect the matrix entries.

2.3.3.2. Algorithm

The general algorithm for corotational elasticity with implicit time integration is described in detail in Section 2.1. In order to update the state of the system, the ODE

$$\mathbf{M}\ddot{\mathbf{u}} + \mathbf{K}\mathbf{u} = \mathbf{f}_{\text{ext}} \tag{2.110}$$

is integrated by applying implicit time integration. This results in the linear system

$$\left(\mathbf{M} + \Delta t^2\mathbf{K}\right)\Delta\dot{\mathbf{u}} = \Delta t\left(\mathbf{f}_{(i)} - \Delta t\mathbf{K}\dot{\mathbf{u}}_{(i)}\right), \tag{2.111}$$

which is solved for changes of the velocities.

The term $\mathbf{Ku}$ in Eq. (2.110) accounts for the internal elastic forces. With Cauchy strain, the relation between displacements and forces is linear in contrast to the use of the Green strain tensor. The artifacts arising from the linearization can be avoided by determining local rotations $\mathbf{R}$ of the deformed body and modifying the computation of the forces accordingly. Etzmuss et al. [EKS03] determined the local rotation per element for cloth simulation, whereas Hauth et al. [HS04] and Müller et al. [MG04] transfer that approach to deformable bodies. A polar decomposition is used to extract the rotational part $\mathbf{R}$ of the deformation gradient $\mathbf{F}|_{\mathbb{T}}$. The current state is then transformed to a non-rotated coordinate system by $\mathbf{R}^T$, the inverse of $\mathbf{R}$. With $\mathbf{u} = \varphi - \mathbf{x}$ the corotational force is

$$\mathbf{f}_{\text{cor}} = \mathbf{R}\left[\mathbf{K}_{\mathbb{T}}(\mathbf{R}^T\varphi - \mathbf{x})\right] = \mathbf{K}'_{\mathbb{T}}\varphi - \mathbf{f}_0, \tag{2.112}$$

where $\mathbf{f}_0 = \mathbf{R}\mathbf{K}_{\mathbb{T}}\mathbf{x}$. Therefore, the global stiffness matrix $\mathbf{K}'$ has to be reconstructed in each time step. However, simply using a polar decomposition for $\mathbf{R}$ can result in inverted elements. Induced by user interaction or heavy collisions, vertices are pushed into other tetrahedra. As this also represents an equilibrium configuration, elements remain in this state.

2.3.3.3. Inversion and corotation of quadratic elements

The most time consuming part in recent publications is the construction or reassembly of the stiffness matrix [MTPS08, PO09]. It consists of extracting the rotational part of the deformation gradient and updating the

affected entries in the stiffness matrix. In order to reduce the time for this crucial step, a specialized traversal (see Section 2.3.4) is developed. Furthermore, an approximation for determining $\mathbf{R}$ with quadratic elements is proposed, which reduces the number of required rotation extractions. Standard FE approaches require computation of integrals by evaluating the functions at predefined integration points. Therefore, Mezger et al. [MTPS08] propose to evaluate and to decompose the deformation gradient at the four integration points that are required for quadratic polynomials. In contrast, the BB-form allows for an analytic computation of the integral and an approximation of the rotation can be applied. Therefore, only the corner control points $\mathbf{b}_{2000}, \mathbf{b}_{0200}, \mathbf{b}_{0020}$ and $\mathbf{b}_{0002}$ are considered to determine the deformation gradient and analyze the element's deformation. This is equivalent to considering only the linear part of the field of displacements. Hence, an acceleration of factor four compared to [MTPS08] is achieved, as only one instead of four rotations per element are extracted. Using the aforementioned approximation Eq. (2.52) reduces to

$$\mathbf{F} = \mathbb{I} + \left(\hat{\mathbf{b}}_{1000} \nabla\lambda_0 + \ldots + \hat{\mathbf{b}}_{0001} \nabla\lambda_3\right),$$

with $\hat{\mathbf{b}}_{1000} = \mathbf{b}_{2000}, \ldots, \hat{\mathbf{b}}_{0001} = \mathbf{b}_{0002}$ and precomputed coefficients $\nabla\lambda_c = \left(\frac{\partial\lambda_c}{\partial x} \frac{\partial\lambda_c}{\partial y} \frac{\partial\lambda_c}{\partial z}\right)$. In the case of an inverted element, a simple polar decomposition will lead to a rotation $\mathbf{R}$ with a negative determinant. This represents a reflection rather than a rotation and will result in implausible behavior. To recover from this undesired state the method proposed by Irving et al. [ITF04] is applied, where an SVD is used to compute $\mathbf{R}$.

2.3.4. Mesh-based matrix assembly and traversal

For the representation of the global sparse matrices, a data structure with *block compressed sparse row storage* (BCSR) similar to Parker et al. [PO09] is used. In this format, every non-diagonal, non-zero entry together with its column index is stored in arrays. The block size 3×3 corresponds to single node-node contributions (see Eq. (2.107)), so the pattern of non-zero entries is a minimal representation of the matrix' structure. Each entry in the global matrix can be made up of several contributions of different elements. As the stiffness matrix has to be updated in every iteration, an efficient traversal scheme is crucial.

The common procedure in standard FE literature for construction of the global matrices consists of two steps. The first part loops over all elements and computes the local contributions. Afterwards, these element matrices are inserted into the global matrix. When constructing or reassembling the matrices with this kind of traversal, the rows have to be inspected to find the entry with matching column index. This can be accelerated by a binary search, if the row entries are sorted. However, in order to avoid searching the rows, the goal is to compute one entry completely and only add it to the matrix data structure once.

In the context of linear FEs, Peña Serna et al. [PnSdSSM09] analyzed the properties of matrices arising from a tetrahedral mesh discretization. The pattern of non-zero entries is directly linked to the topology of the mesh, as every non-diagonal, non-zero entry corresponds to an edge between two vertices. In order to accelerate the assembly for linear elements, Georgii et al. [GW08] propose to store a reference to the corresponding block in the matrix for each edge.

In the case of quadratic FEs, the relationship between topology and entries in the matrix is more complex. Here, each pair of two nodes $d_i, d_j \in \mathcal{D}_\triangle$ that shares at least one tetrahedron correspond to a non-zero matrix entry. These relationships can be further classified into four cases, defined by the lowest dimensional mesh element spanned by d_i and d_j (see Fig. 2.17). One can distinguish, if both nodes are

a) associated with the same vertex $V \in \mathcal{V}$
b) located on the same edge $E \in \mathcal{E}$
c) belong to the same face $F \in \mathcal{F}$

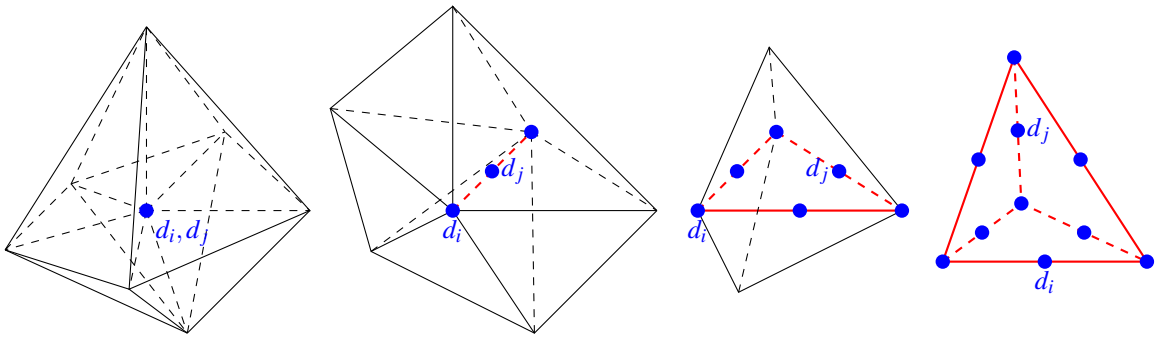


Figure 2.17.: The possible element contributions of two nodes d_i and d_j. The spanned topological elements are shown in red. (a) Two nodes meet at a vertex $\mathcal{V}$. (b) Two nodes are located on the same edge $\mathcal{E}$. (c) Two nodes belong to a face $\mathcal{F}$. (d) Two nodes share a tetrahedron $\mathbb{T}$.

d) share one tetrahedron $\mathbb{T} \in \mathcal{T}$

This distinction also defines how many element contributions accumulate to the final result. While in the first case it is the degree of the vertex V, in the second case it corresponds to the number of tetrahedra adjacent to the edge E. The number of element contributions in the third case is two or one for inner and boundary faces, respectively, whereas in the fourth case only one element matrix contributes. This scheme can be extended easily to FEs with even higher order to reveal the number of contributing element entries. Note that linear FEs are a special case of that scheme, where only the first and second case can occur, leading to the algorithm proposed by Georgii et al. [GW08].

For the fast matrix assembly algorithm this classification has to be precomputed. Therefore, the relevant dependencies between the nodes are identified, i.e., all pairs of nodes d_i and d_j with $i < j$ in conjunction with the spanned topological element $\mathfrak{t} \in \triangle$ are determined. This information is collected by traversing a data structure for tetrahedral meshes with adjacency information (for example the one presented by Peña Serna et al. [PnS-dSSM09]) and storing the connectivity sorted by the index i. The dependencies are then represented by a mapping $d_i \rightarrow (d_j, \mathfrak{t})$ and are used to generate global accumulated matrices with the algorithm given in Listing 2.3.

```
1 def assembly():
2     for i = 0,...,N-1 with d_i ∈ D_△
3         for j with d_j ∈ (d_i → (d_j, t)), t ∈ △
4             for T ∋ t, with t ∈ △, T ∈ T
5                 K_{i,j} += K_{T_(I,J)}
```

Listing 2.3.: Matrix assembly

The algorithm can be considered to be directly related to the respective structure of the matrix. The outer loop corresponds to the rows in the global stiffness matrix or to each node d_i. The second loop is related to all non-zero columns d_j in the matrix, i.e., all nodes d_j that depend on d_i are visited. The inner loop iterates over all element stiffness matrices $\mathbf{K}_{\mathbb{T}_{(I,J)}}$ that contribute to the global entry. Therefore, all relevant tetrahedra are given

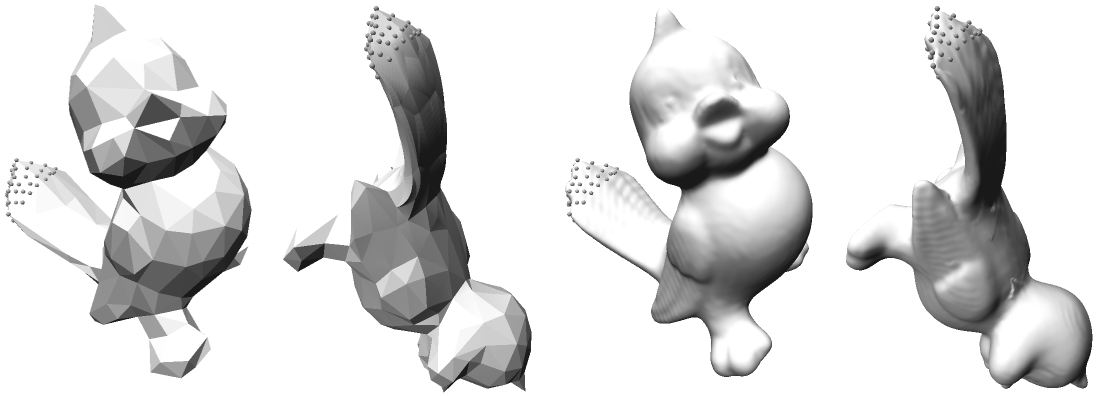


Figure 2.18.: Coupling a high resolution triangle mesh with a coarse simulation mesh (315 tetrahedra). The Tweety model is fixed at the tail and influenced by gravity.

by the adjacency to the topological element $\mathfrak{t}$ (see Fig. 2.17). The presented algorithm can be easily extended to higher-order FEs in order to achieve an assembly per global matrix and not per element entry.

2.3.5. Coupling

In this section, a method is described to couple a high resolution surface triangle mesh with quadratic BB-form bases defined on a volumetric tetrahedral mesh. This yields visually high-quality results and is straightforward to implement with the field of displacements in BB-form.

As in [MG04], for each vertex v_i of the fine triangle mesh the closest tetrahedron $\mathbb{T}_j$ and its barycentric coordinates λ_k are determined in a preprocessing step. The mapping between v_i and $\mathbb{T}_j$ with associated barycentric coordinates λ_k is stored. These relative coordinates are used to interpolate the coordinates during animation. The current displacement of a vertex v_i can then be determined by means of the piecewise quadratic field of displacements in Eq. (2.48) on $\mathbb{T}$. The barycentric coordinates serve as direct input for evaluating polynomials in BB-form with the well-known *algorithm of de Casteljau*

$$\mathbf{b}_{ijkl}^{[\ell]} = \lambda_0 \mathbf{b}_{i+1jkl}^{[\ell-1]} + \lambda_1 \mathbf{b}_{ij+1kl}^{[\ell-1]} + \lambda_2 \mathbf{b}_{ijk+1l}^{[\ell-1]} + \lambda_3 \mathbf{b}_{ijkl+1}^{[\ell-1]},$$

for $\ell = 1,\ldots,2$, with $\mathbf{b}_{ijkl}^{[0]} = \mathbf{b}_{ijkl}$. The result $\mathbf{b}_{0000}^{[2]}$ is the transformed position of the vertex v_j. In order to avoid recomputing the vertex normals of the surface mesh after each simulation step, the rotation matrices of the associated tetrahedra are used. Fig. 2.18 shows the coupling in two different states of the deformation.

Model	Tetrahedra	Nodes	t_{rot}	t_{ele}	t_{mat}	t_{f_0}	t_{sol}	t_{all}
Mechanical element	34446	6896	81	45	49	70	58	304
Bunny huge	9997	2087	22	13	13	20	16	85
Armadillo high	3159	1016	8	4	4	7	10	33

Table 2.2.: Timings for corotational linear FEs (in ms). The major part of the simulation step is used for the corotational formulation, which includes the extraction of rotations (t_{rot}), application of the elements rotations (t_{ele}), assembly of the global stiffness matrix (t_{mat}) and computation of rotated forces t_{f_0}.

Model	Tetrahedra	Nodes	t_{rot}	t_{ele}	t_{mat}	t_{f_0}	t_{sol}	t_{upd}	t_{all}
Armadillo linear	9715	2304	22	12	13	20	18	3	89
Armadillo quadratic	1112	2238	3	16	7	3	35	2	68
Bunny Linear	3213	960	7	4	3	6	4	5	30
Bunny Quadratic	450	1014	1	7	3	1	5	5	22

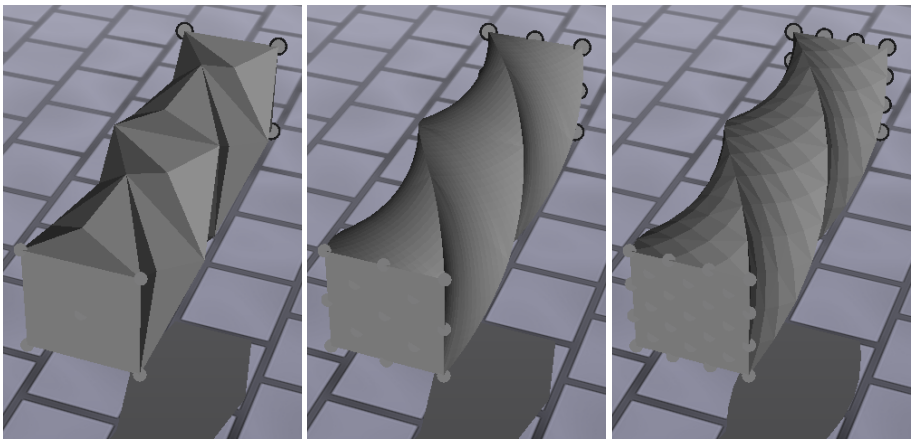
Table 2.3.: Timings for corotational linear and quadratic FEs with comparable complexity w.r.t. the number of nodes (in ms). Simulations with quadratic FEs show higher performance.

2.3.6. Results and discussion

In order to evaluate the proposed methods, a comparison with a simulation using linear corotational FEs is conducted. The computation times for the different parts of the simulation are listed in Table 2.2. All computations were performed on an Intel Core 2 Duo CPU with 3 GHz using only one of the cores. In all cases, a constant time step of $\Delta t = 20$ ms was used, unless specified otherwise. One can see that a considerable amount of effort is spent on the corotational algorithm: Extracting the rotations (t_{rot}), updating the element stiffness matrices (t_{ele}), reconstructing the global stiffness matrix (t_{mat}) and evaluating the additional forces (Eq. (2.112), t_{f_0}) take around 70% to 80% of the simulation time. Recent work that applies the corotational formulation report similar ratios: For linear FEs Georgii et al. [GW08] measured 60% for "Rotate" and "Assemble". Additionally, they must update their mesh hierarchy for the multigrid solver, which requires extra effort in the context of corotation. Parker et al. [PO09] also use about 50% to 60% for the "Solver Setup". In summary, this shows that corotation accounts for the majority of the simulation loop while the linear solver is no longer the bottleneck. In order to reduce this additional effort, the usage of quadratic FEs is proposed, reducing the number of tetrahedra while maintaining quality. Therefore, significantly fewer polar decompositions have to be performed saving a substantial amount of computational effort.

Now, a visual comparison between linear and quadratic FEs is conducted. In Fig. 2.19, a simulation with linear and quadratic FEs with nearly the same number of degrees of freedom (2304 and 2238 nodes for the linear and the quadratic simulation, respectively) is shown. In both cases, the leg of the *Armadillo model* is pulled and pushed. The picture shows the resulting deformation on a coupled high resolution surface. The simulation demonstrates the superior accuracy of quadratic FEs, as more transient detail of the deformation field is apparent. In an animation, small scale deformation at the ears and at the stationary leg of the model that do not occur in the linear case can be identified. The timings presented in Table 2.3 clearly show that with a comparable number of degrees of freedom faster simulations are achieved, as the corotational part is significantly accelerated. But additional time is spent for solving the system of linear equations (for a fair comparison, the same accuracy threshold in both simulations was used). The quadratic simulation requires more iterations for convergence and a single matrix-vector multiplication is more expensive due to the increased number of non-zero entries in the matrix. A further comparison of linear and quadratic FEs is shown in Fig. 2.20, where a bar is exposed to

Figure 2.19.: Top: a linear FE discretization of the *Armadillo model*. Bottom: quadratic FE simulation shows more small scale deformations, e.g., at the ears.



(a) ~100 linear elements (b) ~100 quadratic elements (c) ~2500 linear elements

Figure 2.20.: A bar is subject to torsional movement. (a) Resulting deformation with approximately 100 linear FEs. (b) The same setup with quadratic FEs. (c) A comparable deformation is achieved with significantly more linear FEs, approximately 2500 in this case. The higher-order representation is able to capture this motion more accurately than linear FEs.

Model	Tetrahedra	Nodes	t_{rot}	t_{ele}	t_{mat}	t_{f_0}	t_{solve}	t_{upd}	t_{all}	FPS
Armadillo low	1112	2238	3	16	8 (13)	3	14	2	47	20.3
Armadillo medium	2044	4015	5	30	14 (25)	6	26	2	85	8.1
Armadillo high	3159	6098	7	45	23 (39)	10	44	2	133	5.2
Tweety low	364	757	1	5	2 (4)	2	5	5	19	41.2
Tweety medium	960	1902	2	14	7 (12)	3	12	5	43	20.6
Tweety high	2019	3900	5	29	14 (25)	6	26	5	86	10.8
Bunny low	1006	1973	2	15	7 (12)	3	13	5	46	20.3
Bunny medium	2183	4145	5	31	15 (28)	7	29	5	94	10.3
Bunny high	3213	5949	8	46	24 (40)	10	44	5	138	7.0

Table 2.4.: Different models with varying geometric complexity. The computation time is split into the components t_{rot} (extraction of the rotational component of the deformation per tetrahedron), t_{ele} (updating the element stiffness matrices), t_{mat} (reassembling the stiffness matrix, standard matrix assembly in brackets), t_{solve} (solving the linear system of equations), t_{upd} (updating the fine surface mesh) and t_{all} (total computation time for one frame). The numbers show the timings (in ms) for several stages in the solution process and frames per second, respectively.

torsional deformation. With only a few linear FEs, the features of the deformation cannot be captured (Fig. 2.20 (a)), where the same tetrahedral discretization with quadratic FEs shows more realistic motion (Fig. 2.20 (b)). In order to achieve a comparable simulation with linear FEs, the number of elements must be increased significantly (Fig. 2.20 (c)). In this specific case, the simulation with the few quadratic elements is faster by a factor of five in comparison to the discretization with many linear elements.

Now, the *mesh based matrix assembly method* proposed in Section 2.3.4 is compared with the conventional one. In the standard case, a loop over all element matrices is performed and the contributions are added in arbitrary order. There, a binary search is used to insert entries into the correct column of the global matrix. In Table 2.4, the column t_{mat} represents the time for the matrix construction with the novel traversal method. The number in brackets denote the time for the conventional traversal, which is accelerated by 75% on average. Furthermore, the column t_{rot} denotes the time for extracting the rotations. As each element's rotation is approximated by one rotation matrix, the speed is increased by a factor of four compared to [MTPS08].

2.3.6.1. Examples

In this section, simulations of deformations with different models in various scenarios are presented. In these examples the following material properties are used: A high $\nu = 0.45$ for modeling near-incompressible materials, $\rho = 500\,[\text{kg}\cdot\text{m}^{-3}]$ and E is varied from $200\,[\text{kPa}]$ to $1.5\,[\text{MPa}]$. Fig. 2.15 shows a deformation of the Bear model lifted above the plane and released afterwards. To demonstrate the effect of the corotational approach, the tail of the Tweety model is fixed (see Fig. 2.18) and it is deformed under the influence of gravity. Note that even large rotations do not result in artifacts. The recovery from element inversion is demonstrated in Fig. 2.21, where the material resistance is reduced to zero and increased afterwards by modifying the Young's modulus.

2.3.6.2. Performance Analysis

Table 2.4 summarizes the discrete steps along with the related computation times in the simulation for several models. The name of each model and its resolution (i.e., the number of tetrahedra and nodes) are given. Minor

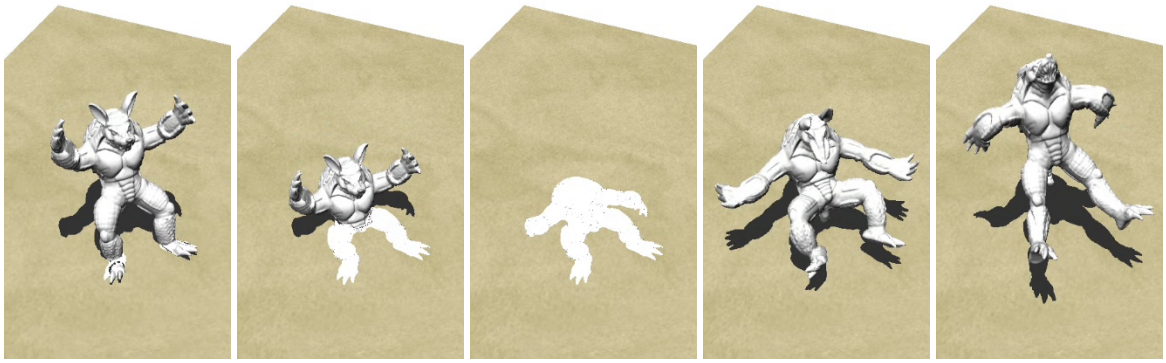


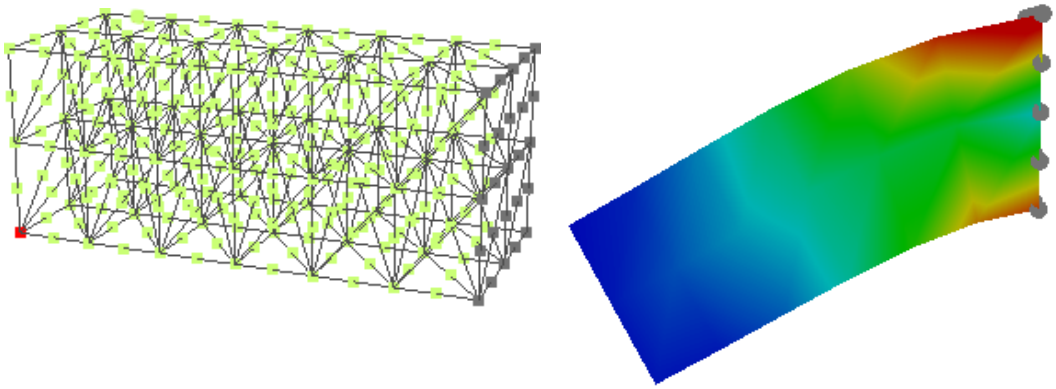
Figure 2.21.: The *Armadillo model* subject to gravity is flattened due to a temporary Young's modulus of zero. After increasing the resistance again the model recovers from the state by overcoming inversions of a lot of elements.

tasks, such as rendering, only slightly affect the simulation and are therefore omitted. The frame rates in Table 2.4 clearly show that simulations are possible in interactive time for small models composed of several hundreds of tetrahedra. For models with about 3000 tetrahedra, a frame rate of 7 fps is still achieved. For example the *Bunny model* with 3213 tetrahedra, which consist of 5949 nodes, a linear system of equations with dimension 17847×17847 must be solved.

In summary, a large number of tetrahedral elements can be simulated at interactive rates. In comparison to Mezger et al. [MTPS08], a performance increase by roughly three times can be observed with about one thousand tetrahedral elements (their *Armadillo 20k* and the *Armadillo low* model presented here). Moreover, due to the reduced number of rotation extractions and efficient reconstruction of the stiffness matrix the proposed approach results in an acceleration. Their corotational formulation requires 73 ms, whereas the proposed method needs about 30 ms ($t_{\text{rot}} + t_{\text{ele}} + t_{\text{mat}} + t_{f_0}$). As no information about the used solver in terms of accuracy and type is available, the timings for solving the linear system are not directly comparable. However, the linear solve requires only 14 ms compared to 66 ms (t_{solve}), which is more than a factor of four.

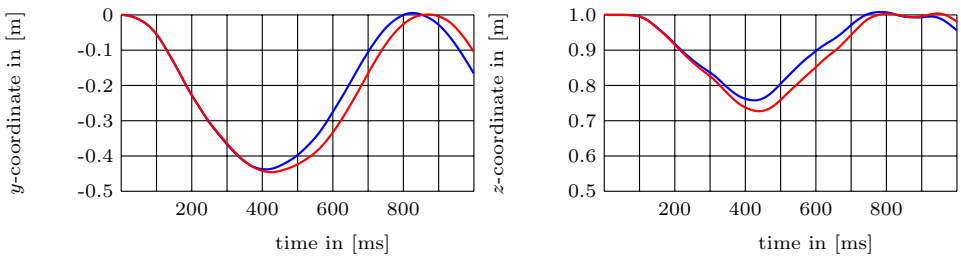
2.3.6.3. Effect of corotation for Quadratic Elements

Now, the Green strain tensor formulation is compared with the corotational approach in BB-form. To avoid computing the time-dependent Jacobian $\nabla_{\mathbf{u}}\mathbf{K}(\mathbf{u})$ for the non-linear formulation, the analysis is restricted to an explicit time integration scheme. As explicit schemes are not unconditionally stable, a time step of $\Delta t = 0.5$ ms is used to achieve a stable simulation for the following scenario. A bar with dimensions $0.4 \times 0.4 \times 1 [m^3]$, Young's modulus $E = 200\,[\text{kPa}]$, Poisson coefficient $\nu = 0.34$ and density $\rho = 500\,\left[\text{kg} \cdot \text{m}^{-3}\right]$ is held fixed on one side and gravity is applied (see Fig. 2.22). The movement of the lower left corner of the bar is tracked. Fig. 2.23 shows the plots of the vertex motion in the corotational (shown in blue) and non-linear approach (red). As one can see, the trajectories are very close, supporting the use of one rotation per element. Only small numerical errors arise from the approximation of the corotation in BB-form (see the detailed discussion in [HS04]).



(a) (b)

Figure 2.22.: (a) A discretized bar with tracked vertex marked in red. (b) Maximum deflection colored with the magnitude of *von Mises stress* from low in blue to high in red.



(a) (b)

Figure 2.23.: (a) Positional changes of the y-component of the lower left corner of the bar. (b) Positional change of the z-component. The blue curve shows the trajectory for the corotational formulation, the red curve the non-linear formulation. The trajectories are close and show that an approximation with one rotation matrix is sufficient.

2.3.7. Summary

In this section, a set of novel methods for dynamic volumetric deformation has been presented. The methods extend previous work on BB-form-based quadratic simulations to large deformations using a co-rotational approach without compromising the simplicity of the mathematical formulation. *The quadratic BB-form method* allows to utilize significantly fewer elements to reproduce complex deformation. For specific cases, the usage results in an acceleration by a factor of four to five. The application of quadratic FEs allows to reduce the additional effort of the corotational method. Furthermore, the use of the BB-form allows to approximate the rotation per element, as the integration does not have to be evaluated at specified integration points. Furthermore, the *mesh-based matrix assembly* to speed up the reconstruction of the matrices for quadratic FEs was presented. There is a significant performance increase when adopting these methods.

The next section covers the extension to the *cubic BB-form method*. It will be shown that a further increase of the order of the basis functions results in more accurate simulations and further reduces the number of elements required. In addition, the *p-multigrid method* will be introduced, which makes use of hierarchies of polynomials in BB-form to solve the corresponding linear systems more quickly.

2.4. The cubic BB-form method and p-multigrid algorithms

In the following, the simulation of volumetric deformation presented in Section 2.3 is extended to cubic basis functions. Furthermore, a novel p-multigrid method for efficient simulation of corotational elasticity with higher-order FEs is presented. In contrast to other multigrid methods proposed for volumetric deformation, the resolution hierarchy is realized with varying polynomial degrees on a tetrahedral mesh. The multigrid approach can be either used as a direct method or as a preconditioner for a CG algorithm. The efficiency of this approach is demonstrated and compared to commonly used direct sparse solvers and PCG methods. As the polynomial representation is defined w.r.t. the same mesh, the update of the matrix hierarchy necessary for corotational elasticity can be computed efficiently. The applicability of cubic FEs for deformation simulation is demonstrated by comparing analytical results in a static setting and scenarios of dynamic simulations with quadratic and cubic elements are shown. Applying the p-multigrid method to quadratic and cubic FEs results in an acceleration of up to a factor of seven for the solution of the linear system. This section is based on the publications [WMRA*14] and [WMRA*15]. Large parts of the text are copied verbatim with minor additions and corrections. Furthermore, pictures and diagrams are reused as well.

2.4.1. Introduction

FEM is a popular choice for simulating volumetric deformation when intuitive material parameters and accuracy are important, as these methods are based on continuum mechanics and do not require parameter tuning to achieve physically plausible results. Instead of the commonly used linear basis functions, some authors propose to simulate volumetric deformation on tetrahedral meshes using quadratic FEs, i.e., using polynomials of degree two (see, e.g., the work of Mezger et al. [MS06] or recently Bargteil et al. [BC14]). This leads to improved accuracy as higher polynomial degrees can better approximate the solution of the PDE. Even though the number of degrees of freedom per element increases, the computational time can be reduced as the desired deformation can be represented with significantly fewer elements.

However, simulating deformation with FEs is computationally expensive, as for a stable simulation, large, sparse linear systems must be solved in every time step. Typically, direct solvers such as sparse Cholesky factorization or the CG method with preconditioning are chosen to solve these linear systems. These are usually the bottleneck of the simulation algorithm and become even more critical with increasing model sizes. A frequent compromise is to use a fixed number of CG iterations, but this in turn increases the numerical damping of the

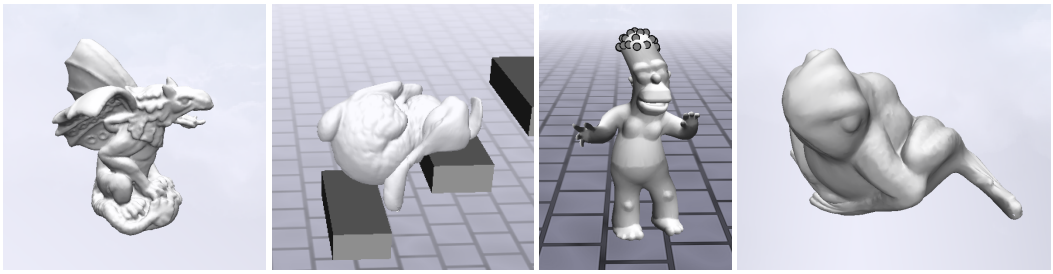


Figure 2.24.: The pictures show simulations using higher-order FEs with the p-multigrid algorithm. From left to right: animation of a *Gargoyle model* using quadratic FE, a *Bunny model* with quadratic FE colliding with obstacles, a toy is animated by changing its boundary conditions (cubic FE) and a *Frog model* with twisted deformation (cubic FE).

simulation and dissipates energy as will be demonstrated. This section addresses this issue and a geometric p-multigrid method is proposed to efficiently and accurately solve sparse linear systems arising from higher-order FEs. The adoption of the p-multigrid method as a preconditioner for a CG method is demonstrated and analyzed. Furthermore, this purely geometric multigrid approach is compared with standard Galerkin coarsening. Additionally, the effect of approximating the field of rotations with only a single rotation matrix per cubic FE is investigated.

In the p-multigrid approach, a hierarchy of varying degree polynomials is constructed to represent the field of unknowns. These are defined on the same tetrahedral discretization to iteratively improve the solution of the linear system. In contrast to other multigrid approaches, these levels vary in polynomial degree instead of mesh resolution. Furthermore, a volumetric deformation simulation technique using cubic FEs on tetrahedral meshes is introduced, which has not been proposed so far for the simulation of time-dependent volumetric deformation. The shape functions are represented as polynomials in BB-form and it is shown how restriction and prolongation of shape functions can be incorporated into the polynomial hierarchy.

The contributions are as follows:

- A novel multigrid solver for volumetric deformation with higher-order FEs is introduced
- Deformation simulations with cubic FEs in BB-form are presented
- It is shown how polynomial representations in BB-form of different degrees can be efficiently transformed into each other
- A speedup for higher-order simulations by up to a factor of seven for solving the linear system in comparison to a PCG method is demonstrated

2.4.2. Higher-order finite element discretization of elasticity

In this section, the general approach for simulating volumetric deformation using higher-order FEs is briefly outlined. At first, the general process of higher-order discretization is described and then the steps necessary for corotational elasticity with cubic FEs are outlined.

2.4.2.1. Finite Element Discretization

The general FE discretization has been described in Section 2.1.2. For completeness, the key points are repeated here briefly and are then explicitly specialized for cubic BB-form polynomials. Given a tetrahedral mesh $\triangle := (\mathcal{T}, \mathcal{F}, \mathcal{E}, \mathcal{V})$ with a set of tetrahedra $\mathcal{T}$, faces $\mathcal{F}$, edges $\mathcal{E}$ and vertices $\mathcal{V}$ approximating the simulation domain Ω, a FE discretization is applied by constructing a piecewise polynomial representation of the displacement field $\mathbf{u} : \Omega \to \mathbb{R}^3$. A degree p polynomial w.r.t. a tetrahedron $\mathbb{T} = [\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3]$ with vertices $\mathbf{p}_i$ is defined by a fixed number of values at nodes. Depending on the order p there are $\binom{p+3}{3}$ degrees of freedom associated with one tetrahedral element at the nodes ξ_I. The positions of the nodes are given by linear combinations of the vertices $\mathbf{p}_i$ and can be determined by

$$\xi_I = \xi^{\mathbb{T}}_{ijkl} = \frac{i\mathbf{p}_0 + j\mathbf{p}_1 + k\mathbf{p}_2 + l\mathbf{p}_3}{p}, \quad |I| = i + j + k + l = p.$$

Here, capital letters are used for a multi-index to simplify the notation, e.g., I represents the set of non-negative indices i, j, k, l. Figure 2.25 shows an example of the lexicographical node numbering for a cubic polynomial defined on one tetrahedral element. The nodes ξ_{3000}, ξ_{0300}, ξ_{0030} and ξ_{0003} are associated with the vertices $\mathbf{p}_0$, $\mathbf{p}_1$, $\mathbf{p}_2$ and $\mathbf{p}_3$, whereas the other node positions are located on the tetrahedron's edges and faces.

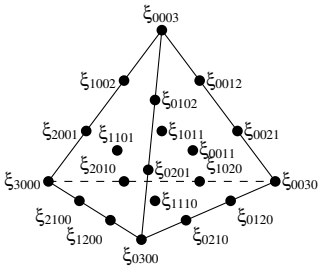


Figure 2.25.: Node numbering for a cubic Bézier tetrahedron. The nodes at the corners of the element are associated with the vertices, e.g., ξ_{3000} corresponds to $\mathbf{v}_0$.

The field of unknowns is represented by a polynomial in BB-form, i.e., the degree p basis functions are the $\binom{p+3}{3}$ Bernstein polynomials

$$B_I^p(\mathbf{x}) = B_{ijkl}^p(\mathbf{x}) = \frac{p!}{i!j!k!l!}\lambda_0^i(\mathbf{x})\lambda_1^j(\mathbf{x})\lambda_2^k(\mathbf{x})\lambda_3^l(\mathbf{x}).$$

Here, $\lambda_i(\mathbf{x})$ are the barycentric coordinates of the tetrahedral element. Just as for the nodes, the non-negative indices sum up to the polynomial degree $|I| = i+j+k+l = p$ reflecting that one basis function is associated with the node with the same index. A polynomial in BB-form per element $\mathbb{T}$ is then defined by

$$f(\mathbf{x}) = \sum_{|I|=p} b_I B_I^p(\mathbf{x}),$$

where in the general form the unknowns b_I can be scalars, vectors or tensors. A general advantage of modeling the field of displacements with this type of shape functions is that the computation of integrals reduces to a sum of coefficients with binomial weights that are computed from the indices i, j, k and l (see Section 2.3). For FE simulations, usually integrals of products of basis functions have to be computed. The products of two basis functions then form higher-order Bernstein polynomials. As polynomials in BB-form are defined w.r.t. barycentric coordinates the chain rule applies when differentiating:

$$\frac{\partial B_I^p}{\partial x_a} = \sum_{i=0}^{3} \frac{\partial B^p}{\partial \lambda_i}\frac{\partial \lambda_i}{\partial x_a}. \tag{2.113}$$

The coefficients $\frac{\partial \lambda_i}{\partial x_a}$ are constant and can be precomputed as the barycentric coordinates are linear polynomials.

In contrast to discontinuous Galerkin methods [KMBG08], FEs are C^0-continuous across element boundaries. Fewer degrees of freedom are required, as adjacent elements share the nodes. Therefore, the number of nodes m not only depends on the number of tetrahedra and the polynomial order but also on the topology of the mesh. The number of degrees of freedom on a tetrahedral mesh $\triangle$ for a FE discretization of order p can be computed by [LS07]

$$m = |\mathcal{V}| + (p-1)|\mathcal{E}| + \binom{p-1}{2}|\mathcal{F}| + \binom{p-1}{3}|\mathcal{T}|, \tag{2.114}$$

which is $|\mathcal{V}|$ for $p = 1$, $|\mathcal{V}| + |\mathcal{E}|$ for $p = 2$ and $|\mathcal{V}| + 2|\mathcal{E}| + |\mathcal{F}|$ for $p = 3$.

2.4.2.2. Corotational formulation of elasticity with higher-order FEM

The corotational formulation of elasticity and the general discretization has been described in Sections 2.1.1 and 2.1.2, respectively. This results in the linear force-displacement relationship

$$\mathbf{K}\mathbf{u} = \mathbf{f}, \tag{2.115}$$

where $\mathbf{K}$ is the stiffness matrix and $\mathbf{u}$ and $\mathbf{f}$ are concatenated vertex displacements and forces. In the following, the novel discretization of higher-order FEs in BB-form is presented. For a degree-p discretization the initial element stiffness matrices ${}^0\mathbf{K}^p_{\mathbb{T}}$ must be computed for each element $\mathbb{T}$. A total of $\binom{p+3}{3}$ 3×3 block matrices are computed, one for each of the elements $\binom{p+3}{3}$ node pairs. As described in Section 2.3, one block of the stiffness matrix for a pair of nodes I,J is computed by

$$\left[\mathbf{K}^p_{\mathbb{T}_{(I,J)}}\right]_{ab} = \int_{\mathbb{T}} \lambda_L \frac{\partial B^p_I}{\partial x_a}\frac{\partial B^p_J}{\partial x_b} + \mu_L \frac{\partial B^p_I}{\partial x_b}\frac{\partial B^p_J}{\partial x_a} + \mu_L\Big(\sum_{c=0}^{2}\frac{\partial B^p_I}{\partial x_c}\frac{\partial B^p_J}{\partial x_c}\Big)\delta_{ab}\,\mathrm{d}V. \tag{2.116}$$

Here, λ_L and μ_L are the elastic Lamé coefficients.

For a FE discretization with cubic polynomials, Bernstein polynomials of degree three must be inserted in Eq. (2.116). The differentiation of B^3_I and B^3_J then results in quadratic polynomials B^2. Therefore, one entry without considering the chain rule of Eq. (2.113) is

$$\int_{\mathbb{T}} \frac{\partial B^3_I}{\partial \lambda_c}\frac{\partial B^3_J}{\partial \lambda_d} dV = 9G(I,J)\int_{\mathbb{T}} B^4\,dV = \frac{1}{35}G(I,J)V_{\mathbb{T}}.$$

Here, $G(I,J)$ are binomial coefficients that are computed as

$$G(I,J) = \frac{\binom{i+\alpha}{i}\binom{j+\beta}{j}\binom{k+\gamma}{k}\binom{l+\delta}{l}}{\binom{2p}{p}},$$

where I and J are multi-indices for i,j,k,l and $\alpha,\beta,\gamma,\delta$, respectively and $p=3$. The additional coefficients originate from the multiplication of two different basis functions. However, the integration is only dependent on the polynomial degree and results in the same value for all basis functions.

The element matrices $\mathbf{K}^p_{\mathbb{T}}$ then have to be assembled into the global stiffness matrix $\mathbf{K}^p$, where a 3×3 entry is non-zero if the corresponding two degrees of freedom share a tetrahedron. In the case of corotational elasticity, the global and element stiffness matrices must be updated using

$$\mathbf{K}^p_{\mathbb{T}} = \mathbf{R}\,{}^0\mathbf{K}^p_{\mathbb{T}}\,\mathbf{R}^T \tag{2.117}$$

in every time step by determining per element rotations $\mathbf{R}$. Similar to Bargteil et al. [BC14] only the linear part of the deformation gradient is used to compute the polar decomposition. For the polynomial hierarchy the rotation matrix only needs to be computed once per element and can be reused on the other levels.

For a dynamic simulation, the lumped mass matrix $\mathbf{M}^p$ is required. Omitting damping terms, a system of ODEs

$$\mathbf{M}^p\ddot{\mathbf{u}}^p + \mathbf{K}^p\mathbf{u}^p = \mathbf{f}^p \tag{2.118}$$

is set up. Here, $\mathbf{f}^p$ are the forces and the dots are derivatives w.r.t. time, i.e., $\ddot{\mathbf{u}}^p$ is the acceleration of the displacements or equivalently the acceleration of positions. A superscript p is added to denote that this system is additionally parametrized by the polynomial degree. The matrices depend on the degree p and the vectors $\ddot{\mathbf{u}}^p$,

$\mathbf{u}^p$ and $\mathbf{f}^p$ are the combined degrees of freedom and therefore represent the time dependent fields in polynomial BB-form. Applying implicit time integration [BW98], a sparse linear system

$$(\mathbf{M}^p + \Delta t^2 \mathbf{K}^p)\Delta \mathbf{v}^p = \Delta t(\mathbf{f} + \Delta t \mathbf{K}\mathbf{v}) \tag{2.119}$$

is set up. Solving this, the change of $\dot{\mathbf{u}}^p$, $\Delta \mathbf{v}^p$ is computed. Then the new velocities are determined by $\mathbf{v}^p_{n+1} = \mathbf{v}^p_n + \Delta \mathbf{v}^p$ and the new positions $\mathbf{p}^p_{n+1} = \mathbf{p}^p_n + \Delta t \mathbf{v}^p$ at all FE nodes are updated. For brevity, Eq. (2.119) is rewritten as

$$\mathbf{A}^p \mathbf{x}^p = \mathbf{b}^p. \tag{2.120}$$

Solving this system is computationally expensive and takes a large part of the time required for a simulation step. However, it is important to solve this accurately as otherwise additional numerical damping is generated (see Section 2.4.6).

In Section 2.4.3, the novel multigrid approach is presented, which enables to efficiently solve this equation by making use of a polynomial hierarchy with varying degree p on the same tetrahedral mesh. In the p-multigrid approach the matrices in Eq. (2.120) are computed independently by simply discretizing with a different polynomial degree.

2.4.3. The p-multigrid method

```
1  def mgvcycle(x_0, b_0):
2      for l in [0...L):
3          x_l := smooth(A_l, x_l, b_l)
4          r_l := b_l − A_l x_l
5          b_{l+1} := I_l^{l+1} r_l; x_{l+1} := 0
6      x_L := A_L^{-1} b_L
7      for l in (L...0]:
8          x_l := x_l + I_{l+1}^l x_{l+1}
9          x_l := smooth(A_l, x_l, b_l)
10     return x_0
```

Listing 2.4.: Multigrid V-cycle

The basic, non-recursive multigrid V-cycle given in Listing 2.4 consists of five operations: restriction ($\mathbf{I}_l^{l+1}$), prolongation ($\mathbf{I}_{l+1}^l$), SpMV ($\mathbf{A}_l\mathbf{x}_l$), smoothing and the exact solution on the lowest level ($\mathbf{A}_L^{-1}\mathbf{x}_L$).

Usually, level 0 corresponds to the finest grid and level L to the coarsest grid. However, in the p-multigrid method on constant grids presented here, the levels correspond to varying polynomial degrees. The algorithm for computing the restriction and prolongation between the levels is outlined in Section 2.4.4 and is a central part of the p-multigrid algorithm that is distinct from other multigrid approaches. The other procedures are similar to standard multigrid implementations and are briefly described in the following. Further details of standard multigrid theory can be found in standard textbooks (see, e.g., [TS01, BHM00]).

Intergrid Transfer: The intergrid transfer operators $\mathbf{I}_l^{l+1}$ and $\mathbf{I}_{l+1}^l$ perform the mapping between different multigrid levels. In the case of p-multigrid on constant grids, they correspond to transformations between the respective polynomial representations, as described in Section 2.4.4.

Multigrid algorithms require the computation of matrices on all levels. In algebraic multigrid procedures, first the intergrid transfer operators are determined and then the matrices using the Galerkin coarse grid operator are computed, i.e., the matrix for level $l+1$ is computed as $\mathbf{A}_{l+1} = \mathbf{I}_l^{l+1}\mathbf{A}_l\mathbf{I}_{l+1}^l$. This requires multiple expensive SpMM operations. In contrast, here the different resolutions are discretized directly, i.e., with different polynomial degrees, which significantly accelerates the overall solution process.

Smoothing: The result of the prolongated solution on the lower level is a correction term $\mathbf{I}_{l+1}^l\mathbf{x}_{l+1}$ which is added to the current level. Although this correction term improves the solution for the low-frequency part of the solution, a high-frequency error term remains which is removed by smoothing. Analogously, high-frequency components of the residual must be removed before restriction by a similar smoothing operator. In this case, low-frequency components correspond to lower-order polynomial terms and high-frequency components to higher-order polynomial terms. As coupling between the degrees of freedom is approximately isotropic, simple point smoothing such as a fixed number of regular Gauss-Seidel or weighted Jacobi iterations can be used. Five Gauss-Seidel iterations are used as a smoothing operator in the experiments and evaluations described in Section 2.4.6.

Exact Solution: For the exact solution on the coarsest level L with the smallest polynomial degree, a standard PCG solver with an relative residual threshold of 10^{-3} is used.

2.4.4. Polynomial intergrid transfer

As described above, the proposed restriction and prolongation operators perform transformations between different polynomial representations. Therefore, each level l in the multigrid hierarchy corresponds to a polynomial of degree p_l. On the lowest level $p_L = 1$ which corresponds to linear FEs. On the highest level $p_0 = p_s$, where p_s is the polynomial degree of the simulation. In the examples p_s is either 2 or 3.

To apply the prolongation operator it is important that a polynomial of degree p can be exactly represented by a polynomial of degree $p+1$. As the degrees of freedom are modeled using polynomials in BB-form (see Section 2.4.2), a standard degree elevation algorithm can be used. The polynomial transformation can be computed by constructing values at each of the $\binom{(p+1)+3}{3}$ nodes [LS07]:

$$b_{ijkl}^{p+1} = \frac{ib_{i-1jkl}^p + jb_{ij-1kl}^p + kb_{ijk-1l}^p + lb_{ijkl-1}^p}{p+1} \tag{2.121}$$

Here, the superscripts denote the polynomial degree of the respective discretization.

For restriction, a degree reduction is performed which is lossy by definition. Following the approach given by Farin [Far02] for univariate Bézier curves, the method is extended to the tetrahedral case. Eq. (2.121) can be rewritten as a system of linear equations

$$\mathbf{b}^{p+1} = \mathbf{P}\mathbf{b}^p. \tag{2.122}$$

The resulting matrix $\mathbf{P}$ is rectangular with $\binom{(p+1)+3}{3}$ rows and $\binom{p+3}{3}$ columns. The rows corresponding to b_{ijkl}^{p+1} where one of the indices equals $p+1$ (i.e., rows associated with vertices) contain only a single non-zero unit entry at the column corresponding to b_{ijkl}^p where the same index equals p. As the system has more equations than unknowns it must be solved in the least squares sense as given by

$$\mathbf{b}^p = (\mathbf{P}^T\mathbf{P})^{-1}\mathbf{P}^T\mathbf{b}^{p+1} = \mathbf{P}^+\mathbf{b}^{p+1} \tag{2.123}$$

However, this introduces dependencies of the vertex values on the other control points. Due to the C^0 continuity requirement in the FEM, vertices, edges and triangles of neighboring elements must share degrees of freedom.

One option would be to combine the matrices P for each tetrahedron into a large matrix and computing the pseudoinverse of that matrix. Although this matrix is only dependent on the topology and can therefore be precomputed, doing so would adversely affect memory consumption and processing power requirements as sparse but large matrices for the restriction operation have to be stored. Therefore, a local, mesh-based scheme is chosen, i.e., subsimplices are processed in order, i.e., first the vertices which are transferred directly, then the edges, the triangles and finally the tetrahedra (the latter two are only relevant for simulation degrees higher than three which were not evaluated). After computing the vertices, the rows and columns associated with these elements are removed and their contributions are added to the right-hand side of the equation resulting in a modified system

$$\mathbf{b}'^{p+1} = \mathbf{P}'\mathbf{b}'^{p}. \tag{2.124}$$

In the cubic to quadratic and the quadratic to linear cases, an analytical solution can easily be found and is used as described in Section 2.4.5.

```
1  def mgpcg(x_0, b):
2      r := b − Ax_0
3      z := mgvcycle(mgvcycle(0, r), r)
4      p := z
5      κ = z^T r
6      while True:
7          p_A := Ap
8          α := κ / (p^T p_A)
9          x := x + αp
10         r := r − αp_A
11         if converged(): return x
12         z := mgvcycle(mgvcycle(0, r), r)
13         κ* := z^T r
14         β := κ* / κ
15         κ := κ*
16         p := z + βp
```

Listing 2.5.: Multigrid-preconditioned CG algorithm, using two multigrid V-cycles as a preconditioner (see Listing 2.4).

The p-multigrid algorithm can also be applied as a preconditioner for a CG algorithm. Listing 2.5 shows a standard CG algorithm with preconditioning. In line 3 and 12 multigrid cycles are applied to improve the conditioning of the matrix. This is in contrast to diagonal preconditioning, where a multiplication with the inverse of the diagonal is performed. The effect of applying these preconditioners is analyzed in Section 2.4.6.

2.4.5. Algorithm

```
1  def precompute():
2      for all 𝕋:
3          ∂λ_i^𝕋/∂x := calculate_derivatives(𝕋)
4          for p in [1, p_s]:
5              ⁰K_𝕋^p = calculate_stiffness_matrix(𝕋)
6      for p in [1, p_s]:
7          M^p := calculate_mass_matrix()
8
9  def simulation_step():
10     R, S := decompose(F)
11     for p in [1, p_s]:
12         K_𝕋^p := R ⁰K_𝕋^p R^T
13         K^p := assemble_matrix(K_𝕋^p for all 𝕋)
14         A^p := M^p + Δ²K^p
15     f^{p_s} := compute_forces()
16     b^{p_s} := Δt(f^{p_s} + ΔtK^{p_s} v^{p_s})
17     Δv^{p_s} := (A^{p_s})^{-1} b^{p_s} # p−multigrid
18     v^{p_s} := v^{p_s} + Δv^{p_s}
19     x^{p_s} = x^{p_s} + Δt v^p
```

Listing 2.6.: Simulation algorithm for p_s levels. Solving the linear system with the p-multigrid algorithm launches several v-cycles (see Listing 2.4) of depth p_s.

Corotational elasticity: In this section, the algorithm for computing corotational elasticity with higher-order FEs is outlined. Furthermore, implementation and optimization details are provided. Especially, the discretization with cubic FEs is a novel contribution not yet addressed for simulating volumetric deformation. Listing 2.6 shows the algorithm for a degree p_s FE simulation using the p-multigrid approach. In the precomputation phase the initial stiffness matrices ${}^0\mathbf{K}_{\mathbb{T}}^p$ are determined for all polynomial degrees p (see Section 2.4.2). This requires the derivatives of the barycentric coordinates $\frac{\partial \lambda_i^{\mathbb{T}}}{\partial \mathbf{x}}$ that carry the geometric information of each tetrahedron. These can be reused for all polynomial levels as the discretization is performed on the same tetrahedral mesh. Additionally, the global mass matrix is set up for all polynomial degrees during precomputation, as it remains constant throughout the simulation.

To set up the linear system all stiffness matrices on the hierarchy must be updated. Therefore, each element's rotation is determined by polar decomposition, the element stiffness matrices are updated and the global stiffness matrix is assembled. Similar to Bargteil et al. [BC14], only the linear part of the deformation gradient is used to compute an element's rotation. Therefore, the rotation matrices are independent of the polynomial degree and can be used on all levels of the hierarchy. In comparison to other approaches that directly solve the system on the highest level, the p-multigrid method additionally requires an update of the system and stiffness matrices on all lower levels of the hierarchy. The governing PDEs are directly discretized with the corresponding polynomial degrees for the respective level. This avoids the computation of expensive SpMM operations in every time step, which would be required for computing the Galerkin coarse grid operator. The system matrix is then constructed

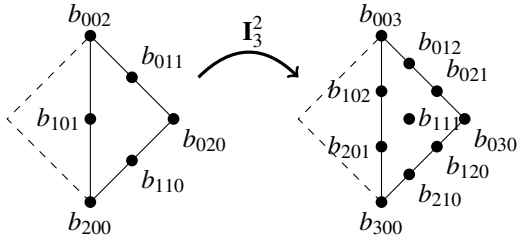


Figure 2.26.: Prolongation operator transferring a field from a quadratic FE to a cubic FE by degree elevation. Note that the degree elevation for edge $(\mathbf{p}_0, \mathbf{p}_2)$ can be reused for both adjacent triangles.

using the precomputed mass matrix and the assembled stiffness matrix. To complete the setup for the linear system, external forces must be determined to compute the right-hand side. Note that this is only required for the highest level of the multigrid hierarchy.

In order to prescribe Dirichlet conditions, a similar approach to the one proposed by Baraff and Witkin [BW98] is applied. They modify the PCG algorithm to filter the respective degrees of freedoms, which is also adopted here for the exact solver. In addition, the Gauss-Seidel iterations are modified to account for the Dirichlet conditions by skipping the updates of constrained vertices. These constraints must be consistent along the multigrid hierarchy, which is easy to achieve as the tetrahedral meshes are identical. However, only entire edges or entire triangular faces are allowed to be constrained in order to avoid special handling of partially constrained values along the hierarchy.

Efficient prolongation: The prolongation operator $\mathbf{I}_{l+1}^{l}$ for transforming polynomials from degree p to $p+1$ can be computed per tetrahedral element using the degree elevation algorithm in Eq. (2.121). However, one can do this more efficiently by taking the C^0-continuity of FEs into account, i.e., some degrees of freedom are shared with adjacent elements. This is illustrated using an example of a prolongation operator applied to a 2D quadratic FE discretization as depicted in Fig. 2.26, where the values b_{ijk}^{p} of the polynomial are associated with the nodes ξ_{ijk}. Additionally, it is exploited that the polynomials in BB-form on a subsimplex, e.g., a vertex, an edge or triangle, can be considered in isolation. In this configuration, some barycentric coordinates are zero and the others vary on the subsimplex and sum to one. Then, the degree elevation can be computed locally on one subsimplex and the other values on the FE can be ignored. Specifically for cubic elements, first the values at the vertices are computed, then at the edges and finally on the triangles.

In the example shown in Fig. 2.26, first the values associated with the vertices of the tetrahedral mesh are transferred directly, e.g., $b_{300}^3 = b_{200}^2$ and only once per vertex. Note that the superscripts denote polynomial degrees instead of an exponent in this context. Then, locally a degree elevation for every edge in the mesh over a 1-dimensional parametric space is performed. In the example, the values along the edge $(\mathbf{p}_0, \mathbf{p}_2)$, i.e., $b_{200}^2, b_{101}^2, b_{002}^2$ can be interpreted as an one-dimensional quadratic Bézier polynomial by simply omitting the second index: $f = b_{20}^2 B_{20}^2(\lambda) + b_{11}^2 B_{11}(\lambda) + b_{02}^2 B_{02}^2(\lambda)$. Then, the standard degree elevation for Bézier curves [Far02] can be applied

$$b_{12}^3 = \frac{1}{3}(b_{20}^2 + 2b_{11}^2), \quad b_{21}^3 = \frac{1}{3}(2b_{11}^2 + b_{02}^2), \tag{2.125}$$

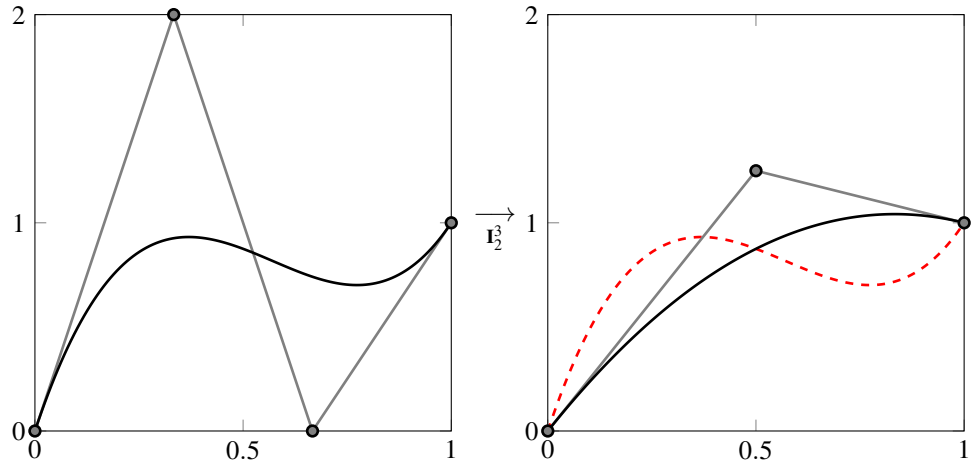


Figure 2.27.: Restriction from cubic to quadratic form using Eq. (2.126). The resulting quadratic polynomial interpolates the value of the cubic polynomial at $t = 0.5$.

where the computation of b_{30}^3 and b_{03}^3 was explicitly omitted. Finally, the remaining value b_{111}^3 at the triangle center is obtained by applying the degree elevation algorithm for Bézier triangles:

$$b_{111}^3 = \frac{1}{3}(b_{110}^2 + b_{101}^2 + b_{011}^2)$$

For volumetric meshes, the degree-elevated values on the edges and the triangles, respectively, can be reused for adjacent tetrahedra.

Efficient restriction: In the case of quadratic and linear polynomials the restriction operator is rather simple. One can simply omit the additional degrees of freedom on the edges, as the vertices are specified directly and no other degrees of freedom exist. For cubic and quadratic polynomials, vertices and edges can be treated separately as done for prolongation. Taking Eq. (2.125) and substituting the vertex values b_{20}^2 and b_{02}^2 by b_{30}^3 and b_{03}^3 respectively, one obtains two equations for b_{11}^2. Taking the average of these values results in

$$b_{11}^2 = \frac{b_{30}^3 + 3b_{21}^3 + 3b_{12}^3 + b_{03}^3}{4} \tag{2.126}$$

and is equivalent to computing $\mathbf{P}'\mathbf{b}'^{p+1}$ and also to computing a quadratic form interpolating the cubic form at $\lambda_0 = \lambda_1 = 0.5$ as shown in Fig. 2.27.

2.4.6. Results

In this subsection, examples are shown and the p-multigrid method is analyzed in terms of performance and accuracy. An Intel Core i7 with 3.4 GHz was used for the tests and all building blocks were implemented utilizing only one of the processor's cores. Table 2.5 lists the models for the tests together with their characteristic numbers, i.e., the polynomial degree, the number of degrees of freedom and number of non-zero entries of the sparse matrix.

Model	degree	# tets	# vertices	# nodes	# dofs	# non-zeros	acceleration
Bunny	2	1691	471	2957	8871	652527	1.72
Gargoyle	2	17040	4490	29218	87654	6544566	2.48
Homer	3	5151	1513	28945	86835	11005101	3.15
Frog	3	7746	2099	42199	126597	16359903	2.74

Table 2.5.: Name, number of tetrahedral elements, number of vertices, number of FE nodes of the models used for the deformation simulation. The last column denotes the acceleration when using the p-multigrid instead of a CG solver.

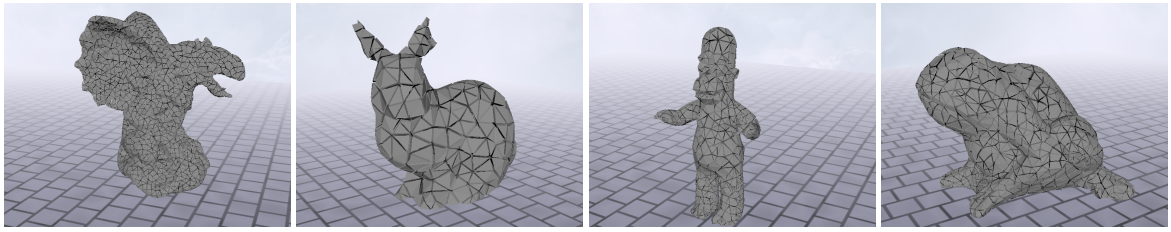


Figure 2.28.: The pictures show the tetrahedral discretization of the models given in Table 2.5. On each tetrahedral element 10 or 20 degrees of freedom for quadratic or cubic elements, respectively, describe the field of displacements. Adjacent degrees of freedom for two or more tetrahedra are shared.

Examples: The snapshots in Fig. 2.24 show different scenarios with quadratic and cubic FEs using the p-multigrid algorithm to solve the linear system. In all cases, the simulation mesh is coupled with a high-resolution surface mesh for rendering. Pictures of the tetrahedral discretization are given in Fig. 2.28.

Validation: To validate the cubic FE approach, a static analysis of a bar fixed on one side deforming under an applied load has been performed. In contrast to the geometries given in Table 2.5, this geometry is well suited to evaluate the accuracy as analytical results are available. A bar with dimensions $(l \times w \times h) = (1.0\text{m} \times 0.2\text{m} \times 0.2\text{m})$ and an elastic modulus of $E = 500$ kPa is chosen. By applying a force of $F = 10$ N at the position in the middle of the free end of the bar, it deforms and the vertical displacement is measured. For the numerical computation, a model consisting of five cubes subdivided in 24 tetrahedra each was constructed that was simulated with linear, quadratic and cubic FEs. As shown in Table 2.6, with a deflection of approximately 0.03 m for linear, 0.048 m for quadratic and 0.05 m for cubic FEs, convergence can be observed when raising the polynomial degree. The computed deflections were compared with an analytical result from beam theory according to the textbook [GHSW07], which is only valid for small deformations. For this model the moment of inertia for the cross-sectional area is $I = \frac{1}{7500}\text{m}^4$. The analytical maximum deflection in vertical direction is then determined by $w_{\max} = \frac{Fl^3}{3EI} = 0.05$ m, which is very close to the computation presented here as it only differs at the fifth decimal place. This shows that cubic FEs can better approximate the solution of the PDE of elasticity. However, it is expected that a further increase of degree of the basis functions will not pay off in terms of accuracy.

Effect of approximate solution of the linear system: In order to speed up the simulation and achieve constant computation times, many authors (see, e.g., [NPF05, ACF11, WBS*13]) propose using a small, fixed number of 20 CG iterations to solve the linear system. This choice leads to artificial damping, as some residual forces

Discretization	linear FEs	quadratic FEs	cubic FEs	Analytic result
Deflection $[m]$	0.03	0.048	0.05	0.05

Table 2.6.: The relative deflection of a bar subject to a vertical force computed linear, quadratic and cubic FEs is compared to an analytic result. The deflection with cubic FEs differs only at the fifth decimal place.

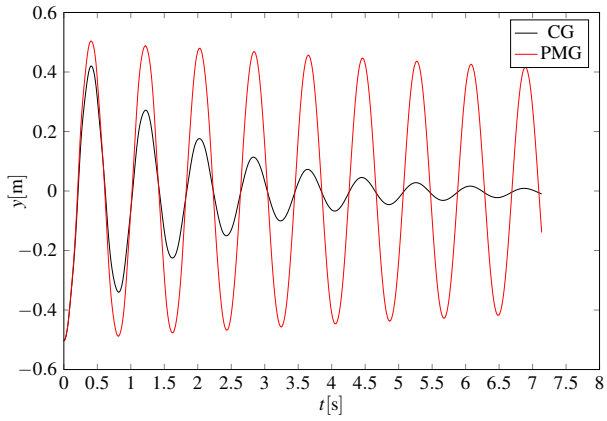
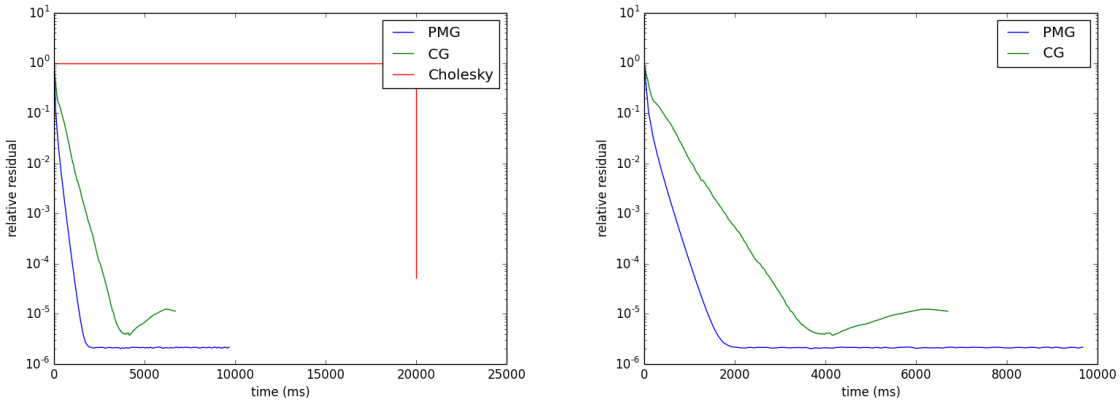


Figure 2.29.: Comparison of the deflection over time of an oscillating bar. The black curve shows the deflection using 20 CG iterations, whereas the red curve is generated when solving accurately with the p-multigrid method.

remain that are not converted into momentum. The following experiment demonstrates this effect. A bar is fixed on one side and initialized to a deflected state. As the deflected side is not fixed, the simulation then causes the bar to oscillate. As no explicit damping terms are applied, the bar should oscillate without energy loss and the amplitude of the oscillation should remain constant. Fig. 2.29 shows the amplitude over time for a CG algorithm with a fixed number of 20 iterations in comparison to a p-multigrid solver with a prescribed residual reduction of 10^{-3}. One can clearly see that the simulation with accurate solves exhibits significantly reduced damping. The remaining damping is caused by the implicit integration scheme used and can be reduced by decreasing Δt or further increasing the iterations. However, a future extension of this approach could adopt an energy-preserving variational integrator [KYT*06] to further reduce damping. Much like Mullen et al. [MCP*09] argument for numerical dissipation in fluid simulations, reducing artificial numerical damping to a minimum is very important to put desired damping under artist control to maximize realism.

Performance: In order to analyze the performance of the p-multigrid solver, the convergence is monitored, i.e., the residuals $\mathbf{r}_i = \mathbf{b} - \mathbf{A}\mathbf{x}_i$ for the i-th iteration. The relative residual reduction ($\|\mathbf{r}_i\|_2 / \|\mathbf{r}_0\|_2$) w.r.t. time of the p-multigrid method is then compared with a PCG algorithm and a direct sparse solver (a Cholesky factorization method [GJ*10]). For the tests, the models in Table 2.5 are compressed along the y-axis and released afterwards. The diagrams in Fig. 2.30 show the relative residual reduction for the model *Gargoyle*. As the direct method finishes without any intermediate results, the graph shows a vertical line. In contrast to the CG algorithm, the residuals of the multigrid algorithm are smooth in the logarithmic plot making it more predictable. The last column in Table 2.5 denotes acceleration when using the p-multigrid instead of a CG solver. Additionally, tests were performed for even larger model sizes. The diagram in Fig. 2.31 (a) shows a plot of the acceleration over the number of nodes (degrees of freedom) for quadratic FEs. This is the expected behavior as the multigrid



(a) (b)

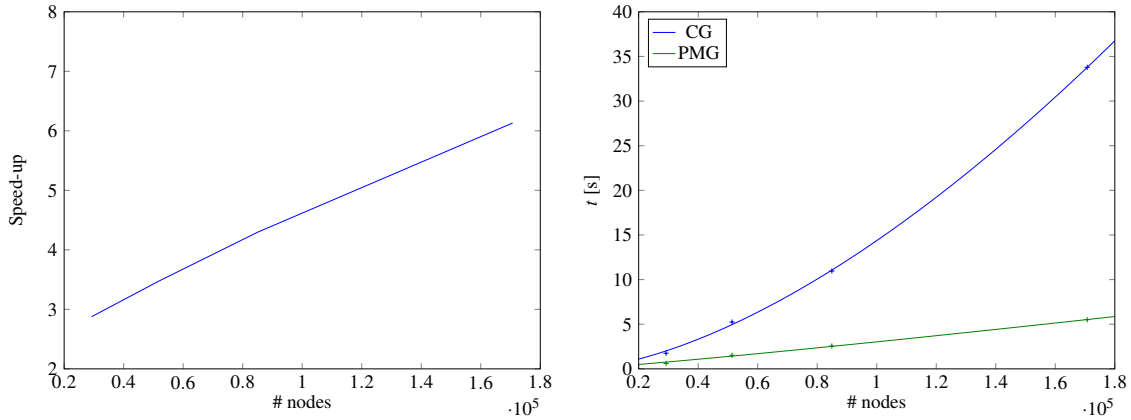
Figure 2.30.: (a) The diagram shows logarithmic plots of the residual over time for the PMG approach, a PCG solver and a direct solver with Cholesky factorization. (b) The Cholesky factorization is omitted to better show the convergence behavior. The multigrid solver reaches a residual of 10^{-3} after 7 iterations. The CG solver requires 135 iterations.

algorithm starts to pay off for large models. Furthermore, the diagram in Fig. 2.31 (b) shows that for a number of second order examples the time to solution for the CG solver increases by approximately $n^{1.6}$ and only by $n^{1.1}$ for the p-multigrid (PMG) solver. These growth rates correspond to a nonlinear least squares fit of a power function an^k to the data points. Determining the complexity analytically is difficult, as it depends not only on the number of degrees of freedom but on the topology of the mesh, i.e., the numbers of vertices, edges and for third order also faces.

Furthermore, two cycles of the p-multigrid algorithm were applied as a preconditioner for a CG solver. A standard PCG algorithm is used, where the preconditioner $\mathbf{M}^{-1}$ is computed by applying two multigrid V-cycles, as shown in Listing 2.5. This hybrid multigrid-preconditioned CG solver (MGPCG) was evaluated on the *Frog model* (see Table 2.5) which was subjected to an initial twisting deformation (see Fig. 2.24). The diagrams in Fig. 2.32 on the left compare the relative residual improvement of the MGPCG solver with the unmodified PMG algorithm based on computation time and number of iterations. For generating the graphs on the right, the *Gargoyle model* using quadratic FEs was compressed along the y-axis and released afterwards. Although the residual improvement per iteration of the MGPCG is better, the increased memory and bandwidth costs lead to reduced performance.

In the *Frog scenario* in Fig. 2.24 with cubic FEs (see Table 2.5 for the mesh complexity), two to three V-cycles are needed to reduce the residual by three orders of magnitude. A single V-cycle in the p-multigrid algorithm takes approximately 235ms on a single core implementation for this example. This splits in approximately 130ms and 72ms for smoothing (each with five Gauss-Seidel iterations) on the cubic and quadratic level, respectively, where the remaining time is needed for restriction, prolongation and the exact solution of the linear system on the lowest level. Note that the exact solution takes only 3ms.

In addition to the computation for solving the linear system, the hierarchy of matrices must be updated. However, this additional overhead compared to non-multigrid methods is compensated due to the improved conver-



(a) (b)

Figure 2.31.: (a) Acceleration of using the p-multigrid solver instead of a CG method with preconditioning for quadratic FEs. The x-axis denotes the number of nodes of the simulation. This demonstrates that multigrid algorithms pay off at large matrix sizes. (b) Time to solution over the number of nodes for both solvers.

gence rate. A fundamental difference besides the polynomial hierarchy and an algorithmic multigrid computing the matrices on linear shape functions (e.g., Georgii et al. [GW06]) is the update of the matrix hierarchy. Whereas their approach requires recomputation of the matrices on each level by two expensive sparse matrix products multiplying the restriction and prolongation operators with the system matrix, the p-multigrid method can directly use the matrices for different polynomial degrees. Even in an optimized version, Georgii et al. [GW08] report that the matrix update takes more time than the multigrid algorithm itself.

In the case presented here, the matrix construction only takes a small percentage of the time required for the solve. For the *Frog example* described above, a computation time of 89ms and 12 ms on average per time step was observed to construct the additional required matrices for $p = 2$ and $p = 1$, respectively. With an average time of 697ms to solve for one time step, these timings relate to 12.8% and 1.7%, respectively. This sums up to a total of 14.5% of time of the entire solve being used to construct the matrix hierarchy in this particular example. This additional effort for constructing matrices on lower levels of the hierarchy splits up into roughly equal parts of the update of the element stiffness matrices and the assembly of the global matrix. Furthermore, it is worth noting that a geometric multigrid approach as proposed by Georgii et al. has not yet been developed for higher-order FEs.

Effect of rotation approximation: Now the approximation of using only a constant rotation per element as opposed to a spatially varying field of rotation is analyzed. For cubic FEs the deformation gradient field is given as a spatially varying quadratic function over each element, as the cubic displacement field is differentiated once. It can also be represented as a polynomial in BB-form with ten degrees of freedom each given by a 3×3 matrix. The rotation field is therefore also spatially varying and ten polar decompositions would be necessary to compute the quadratic polynomial. This field is approximated by using only one rotation matrix, i.e., a constant rotation per element is derived from the linear part of the displacement fields. This is computationally attractive, as in addition to saving nine costly decompositions on the highest level, the rotation matrix can be reused across the polynomial hierarchy and does not need to be recomputed.

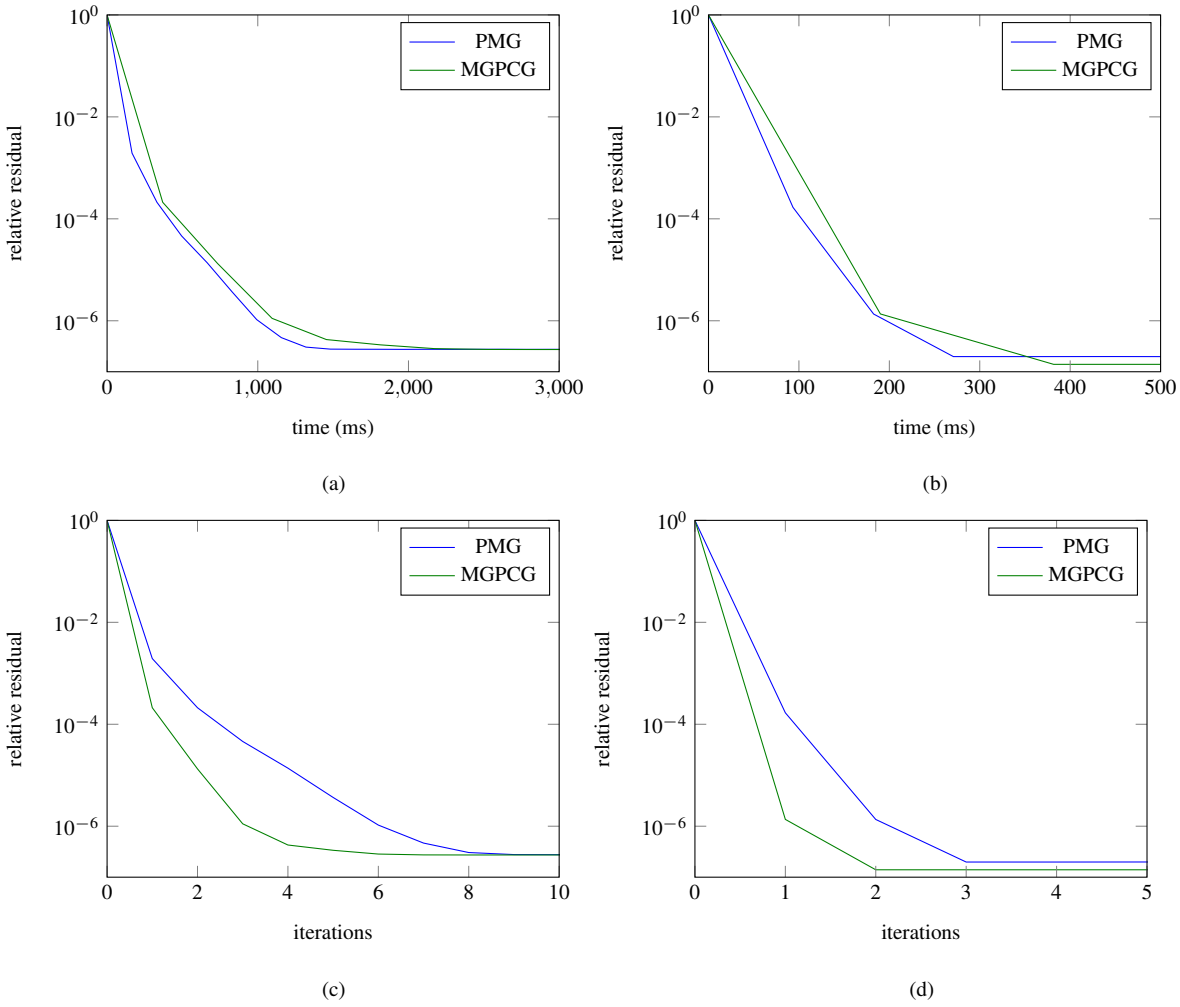


Figure 2.32.: The diagrams compare the PMG algorithm with a multigrid-preconditioned CG solver (MGPCG) using two PMG cycles as a preconditioner with cubic ((a) and (c)) and quadratic FEs ((b) and (d)). The plots (a) and (b) show the residual improvement over time, whereas (c) and (d) depict residual improvement based on the number of iterations. The MGPCG shows better performance per iteration, but is slower when comparing computation time.

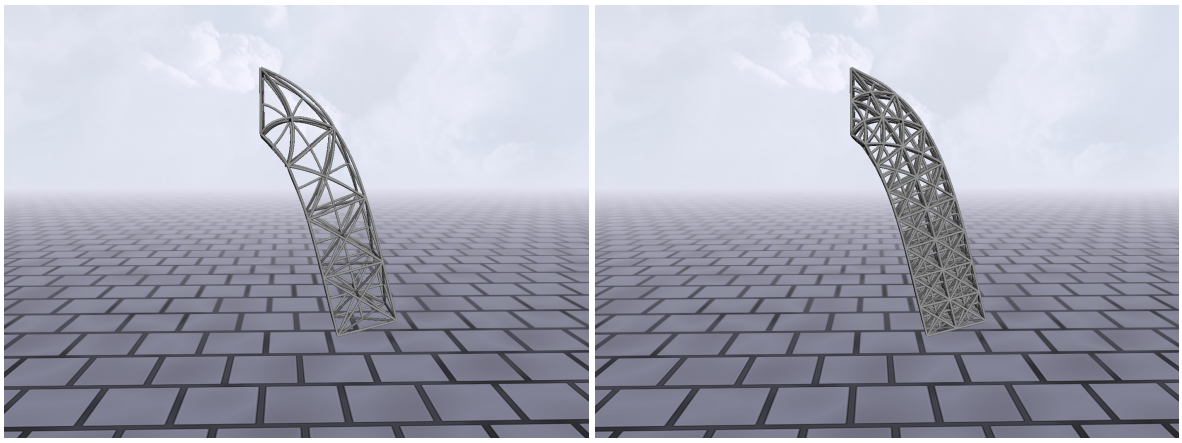
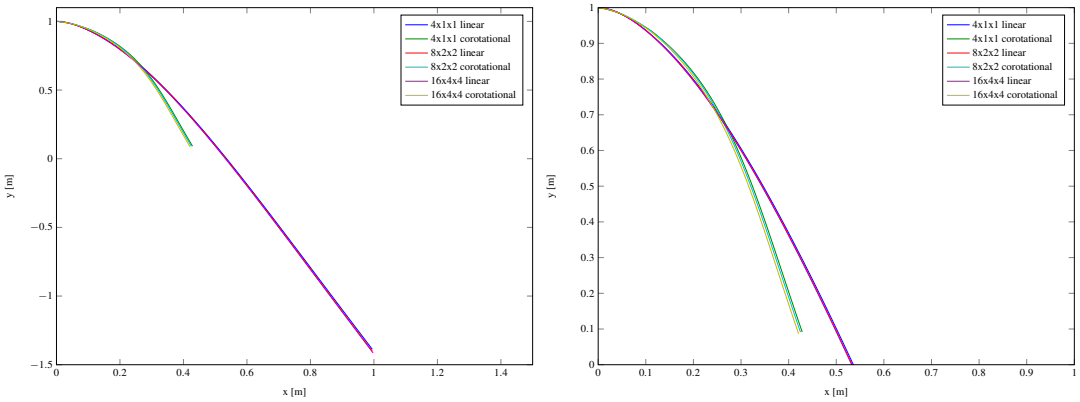


Figure 2.33.: Deformation of a bar under gravity fixed on one side with different discretizations. Here, the discretizations using $4 \times 1 \times 1$ and $8 \times 2 \times 2$ cubes are shown. The number of elements does not significantly influence the resulting deformation.



(a) (b)

Figure 2.34.: (a) The deformation of the centerlines of a bar under gravity is plotted. In (b) an enlarged view of the upper left corner of (a) is shown. The difference of linear and corotational elasticity is apparent by the significant increase of the bar's length for geometric linear modeling. Even with a coarse discretization the corotational approach with constant rotation matrices is able to correct the error induced by linear elasticity.

Discretization	$4 \times 1 \times 1$	$8 \times 2 \times 2$	$16 \times 4 \times 4$
Lin. elasticity $[m]$	2.62	2.63	2.65
Corot. elasticity $[m]$	1.04	1.04	1.04

Table 2.7.: The length of the center line (see Fig. 2.34) of a deformed bar for linear and corotational elasticity and different discretizations is listed. As depicted in Fig. 2.33, the deformation is nearly identical.

			MG Cycles			
Multigrid operator	Update	Cycle	10^{-2}	10^{-3}	10^{-4}	10^{-5}
Geometric	0.14 ms	2.0 ms	4.8	6.3	8.3	10.3
Galerkin	10.8 ms	2.8 ms	3.0	4.5	5.8	7.0

Table 2.8.: Coarse grid update times, time per multigrid cycle and the average number of multigrid cycles required to reach a given residual for the geometric and Galerkin coarse grid operators.

The approximation is based on the assumption that the rotation does not vary significantly within a single element and that a constant rotation field is sufficient for visual plausibility. In order to verify this assumption, different discretizations of the bar from the static validation example were used and the deformation under gravity was visually compared.

The different discretizations are generated by subdividing the bar into $4 \times 1 \times 1$, $8 \times 2 \times 2$ and $16 \times 4 \times 4$ cubes. Each cube is again subdivided into 24 equally sized tetrahedra and a cubic FE discretization is applied. Fig. 2.33 shows snapshots from the simulations. In order to judge the effect of the approximation, the centerlines of the deformed bar are extracted and drawn in Fig. 2.34 for the different discretizations and elasticity models.

One can clearly see artifacts when using linear elasticity as the length of the centerline and the volume (not visible in the diagram) increase non-physically. The error induced by the linear model is compensated in the corotational approach by adapting forces and stiffness matrices to consider the local rotation. Additionally, the length of the center line of the deformed bars are computed and listed in Table 2.7. The deformation of the bar with $16 \times 4 \times 4$ cubes is considered as the reference solution. In the coarsely discretized case one tetrahedral element spans one quarter of the bar, which only uses a single rotation matrix derived from the linear part of the displacement. Even in this case, the error introduced by linear elasticity can be corrected with a constant rotation per element. In conclusion, an approximation using a single rotation matrix is a sufficient approximation, especially for visual plausibility.

Comparison with Galerkin coarse grid operators: Galerkin coarse grid operators are more compatible with the used restriction and prolongation operators $\mathbf{I}_l^{l+1}$ and $\mathbf{I}_{l+1}^l$, as the coarse grid matrices are computed from $\mathbf{A}^{l+1} = \mathbf{I}_l^{l+1}\mathbf{A}^l\mathbf{I}_{l+1}^l$. This typically leads to better convergence than the purely geometric approach. However, this requires two expensive sparse matrix-matrix products. To evaluate if this increased cost is amortized by the reduced number of multigrid cycles, a comparison of the purely geometric method with Galerkin coarse grid operators is conducted. The scenario from the previous section ($4 \times 1 \times 1$ bar using cubic elements) was used. The results, with varying threshold for solving the linear system, are shown in Table 2.8. Using Galerkin coarse grid operators, the update of the multigrid hierarchy takes an average of 10.8 ms, nearly 80 times the time taken for the geometric approach. On average, the Galerkin approach requires 30% fewer multigrid cycles. However, this is not enough to amortize the cost of the multigrid update which must be performed for every simulation step and the more expensive multigrid cycle due to the different sparsity pattern.

2.4.7. Summary

In this section, the *p-multigrid method* for efficiently solving sparse linear systems arising from higher-order FE discretizations has been presented. Due to the direct discretization of the problem with different polynomial degrees on the same tetrahedral mesh, updates of the matrix hierarchy do not have to be computed by expensive SpMM operations. Furthermore, the use of the *cubic BB-form method* for simulating volumetric deformation was demonstrated.

As the depth of the multigrid hierarchy is limited to the polynomial degree, linear scale-up can only be achieved for increasing polynomial degrees, not for models with a large number of tetrahedra. Therefore, it would be interesting to investigate if a geometric multigrid approach on the lowest level would be beneficial for even faster convergence. However, geometric multigrid approaches are not trivially applicable to unstructured grids. As the *p-multigrid method* should be applicable to hexahedral elements as well, it could be interesting to try using block structured meshes as geometric multigrid methods are easy to implement for such meshes. Parallelizing all of the building blocks of the proposed multigrid algorithm on GPUs and comparing the speedup to the one achieved in Section 2.5 could make the algorithm viable for interactive use. GPU performance benefits might also be in the order comparable to the multigrid work on hexahedral elements by Dick et al. [DGW11a] and [DGW11b].

In the next section, methods and data structures for massively parallel computation are developed. Therefore, the *BIN-BCSR data structure* and the *MCG method* is introduced, which significantly accelerate the computations when executed on GPUs. This further decreases computation time and increases the number of elements that can be simulated at interactive rates.

2.5. GPU data structures and methods for efficient deformation simulation

In this section, GPU data structures and algorithms are presented to efficiently solve sparse linear systems which are typically required in simulations of multibody systems and deformable bodies. Thereby, an efficient sparse matrix data structure is introduced that can handle arbitrary sparsity patterns and outperforms current state-of-the-art GPU implementations for SpMV. A kernel merge approach for a CG algorithm is developed reducing the number of kernel calls and therefore avoiding the associated overhead. Moreover, an efficient method to construct global matrices on the GPU is presented where hundreds of thousands of individual element contributions are assembled in a few milliseconds. The algorithms and data structures are specifically developed to enable fine-grained parallelism for the accelerated simulation of volumetric deformation on GPUs.

A FE-based method for the simulation of deformable solids as well as an impulse-based method for rigid bodies are introduced in order to demonstrate the advantages of the novel data structures and algorithms. These applications share the characteristic that a major computational effort consists of building and solving systems of linear equations in every time step. This solver results in a speedup factor of up to 13 in comparison to other GPU methods. This section is based on the publication [WBS*13]. Large parts of the text are copied verbatim with minor additions and corrections. Furthermore, pictures and diagrams are reused as well.

2.5.1. Introduction

Realistic deformable models often contain stiff components which lead to stiff differential equations. For example, even low elastic coefficients in FE simulations force a limitation of the time step size. Many approaches use an implicit time integration scheme, which is unconditionally stable in order to solve the stability problems with explicit integration methods. Articulated rigid bodies are often simulated using a Lagrange multiplier or an impulse-based method. Hence, most simulation methods for deformable models and articulated rigid bodies must solve a large system of linear equations, where the corresponding system matrices have to be updated in every time step. These linear systems are typically sparse, symmetric and positive definite, so a CG solver can be applied. In general, building and solving linear systems requires a major computational effort in each step. Therefore, a fast solver is an essential component for interactive simulations.

Graphics processing units (GPUs) with their high computational throughput theoretically offer a tremendous speedup by offering the possibility to execute special GPU programs also known as kernels. In the past years, several GPU-based solvers have been presented in order to accelerate the solution of linear systems. For exam-

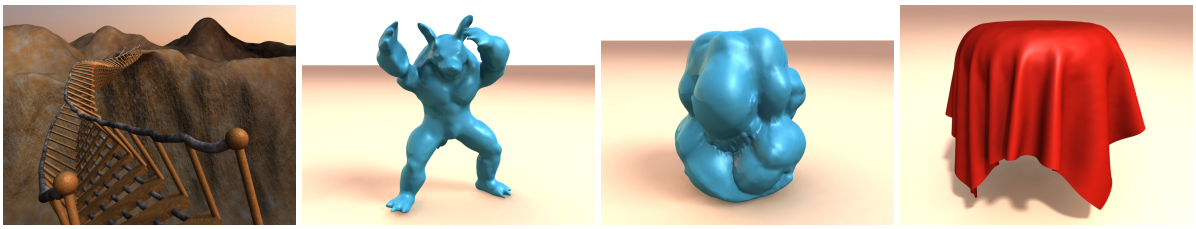


Figure 2.35.: Interactive simulations with the novel GPU-based solver. From left to right: simulations of complex articulated bodies, volumetric deformation with 10K quadratic FEs and 47K degrees of freedom, volumetric deformation with 65K linear FEs and 38K degrees of freedom and highly detailed cloth models.

ple, many fluid dynamics applications in computer graphics can be significantly sped up by those methods, as there the matrix is constant during the simulation. However, for deformable and articulated bodies the matrix changes in each step. Hence, the matrix construction must also run fast to make an efficient simulation possible. Furthermore, it is desirable to be able to control the velocity and position change of certain elements, e.g., for collision handling. Existing GPU solvers provide neither an efficient update of the linear system nor a possibility to control velocities or positions.

In this section, a novel GPU-based solver is presented which allows a fast update of sparse matrix structures and provides velocity and position control. Furthermore, a novel data structure is introduced that accelerates SpMV and therefore the solution of sparse linear systems significantly. In contrast to existing approaches, the goal is to minimize the number of kernel calls needed, as in interactive applications the kernel launch overhead can influence performance significantly. Due to an optimized PCG algorithm, the solver outperforms current state-of-the-art solvers considerably. The presented method permits real-time simulations of very complex deformable and articulated bodies. In order to demonstrate the performance gain in practice, simulations for deformable bodies using linear and quadratic FEs, cloth simulations and simulations of articulated bodies will be presented. Finally, a comparison with current state-of-the-art solvers is conducted and a performance gain by a factor of up roughly 13 is observed.

The contributions are:

- A novel GPU-based PCG algorithm with position and velocity control which is designed for the use in dynamics simulations. It is optimized by minimizing the number of kernel calls.
- A novel GPU data structure for arbitrary sparsity patterns that shows beyond state-of-the-art performance for matrix-vector products.
- An algorithm that efficiently updates sparse matrices on the GPU.

2.5.2. Programming massively parallel hardware

In the following, the concepts of programming massively parallel hardware that serve as a basis for the developed GPU data structures and methods are outlined. In general, there are several options for developing GPU programs. In this thesis, the implementation is conducted with NVIDIA CUDA [NVI15] and restricted to NVIDIA GPUs. However, there are other options such as using OpenCL [SGS10] that make use of other types of massively parallel hardware.

A GPU incorporates thousands of processor cores on a single chip. Therefore, GPUs have a much higher theoretical peak performance in comparison to CPUs. Furthermore, the concept of lightweight thread execution is an important differentiator, which allows for efficient spawning of millions of threads with significantly lower overhead compared to CPU architectures. The GPU is used as a coprocessor where small programs, so-called *kernels*, are launched by a CPU call. By specifying the number of parallel executions, the instructions in the kernels are run in parallel for every thread.

GPU architectures realize high-degree parallelism by executing SIMT operations (single instruction multiple threads). This is similar to the more common SIMD (single instruction multiple data) operations, where the same instructions are executed on different data sets in parallel. The difference is that the threads in SIMT operations may take different paths of execution. However, this should be avoided for threads that are executed concurrently because of performance reasons. Every thread has a unique index that is used to process different chunks of data. On GPUs, processor cores are organized in sets of streaming multiprocessors (SM) as shown in

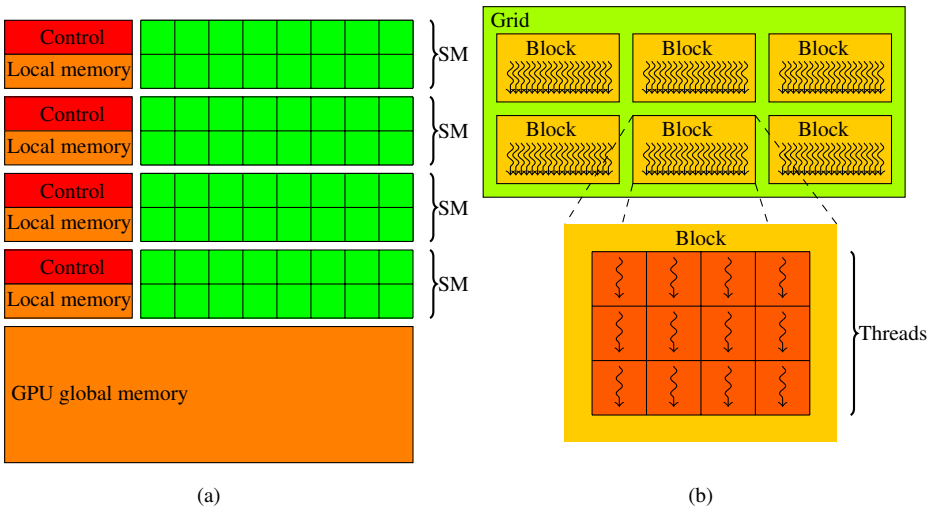


Figure 2.36.: (a) A simplified representation of GPU hardware similar to Fig. 3 in [NVI15]. Processor cores are organized in sets of streaming multiprocessors (SM), which have a control unit and local memory, i.e., registers and shared memory. (b) CUDA thread execution model similar to Fig. 6 in [NVI15]. GPU programs are launched as a grid that is a set of blocks. Each block consists of a set of threads that are executed on a single SM.

Fig. 2.36 (a). Each SM consists of a control unit and local memory like registers and shared memory, where the latter is user-managed cache.

A GPU kernel is launched by specifying the execution configuration as shown in Fig. 2.36 (b). The execution configuration consists of the number of required threads and blocks that make up a grid. Each block is executed on one SM, which ensures that threads associated with the same block can access shared memory. When executing a kernel, the threads of a block are again grouped into warps that consist of 32 threads on current hardware. This atomic unit is then executed exclusively on one single SM in parallel. When computations are finished or an active warp is waiting for a memory transaction, the next warp is scheduled for execution on the SM.

In order to fully exploit massively parallel hardware, the processed data must be sufficiently large. In addition, memory latency has to be compensated to fully utilize all processor cores. As memory operations are usually subject to latency of 400 to 800 clock cycles [NVI15], a sufficient number of warps are necessary to hide latency and to fully exploit the computational power. The memory transactions described in the following are specific to the CUDA execution model. In order to streamline the memory access and make the best use of precious memory bandwidth, the transactions of a warp are coalesced if possible. Therefore, for each warp and kernel the GPU determines the address range of GPU global memory that will be accessed for read or write operations. Afterwards, these memory accesses are combined and issued as coalesced memory transactions that are either 64, 128 or 256 Byte large. The number of required transactions is therefore dependent on the kernels' access patterns. Fig. 2.37 shows a simplified architecture with different types of access patterns. There, all scenarios require loading four units from global memory and coalesced load operations are four units in size in this simplified sketch. Although real examples are more complex and coalesced memory operations are of different size, this example shows that adjacent threads should ideally access adjacent data to minimize the number of memory transactions required. Furthermore, data access should be aligned to minimize the number of transactions.

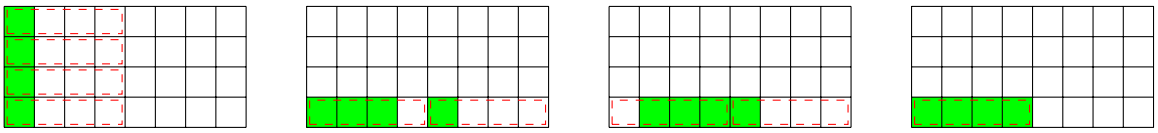


Figure 2.37.: Different types of load operations from GPU global memory for a simplified architecture with a warp size of four and coalesced memory transactions of the same size. Four different examples are shown each loading four units of data highlighted in green. The resulting coalesced load operations are shown as dashed red lines. (a) Memory access with a large stride resulting in four load operations. (b) Non-uniform access resulting in two load operations. (c) Unaligned memory access resulting in two memory operations. (d) Optimally coalesced memory access with a minimal number of memory operations.

In addition to high occupancy and coalesced memory access, there are other important criteria to ensure high performance such as the avoidance of branches or minimizing register usage. However, this is not further discussed here, as the data structures and methods developed in this thesis aim to optimize memory transactions.

2.5.3. BIN-CSR and BIN-BCSR data structure

In this section, new GPU data structures that perform an efficient matrix-vector product

$$\mathbf{y} \leftarrow \mathbf{A}\mathbf{x}$$

are presented. Here, the matrix $\mathbf{A}$ is sparse, i.e., there are only a small number of non-zero entries. Furthermore, the number of non-zero entries per row varies across the matrix. In the simulation applications presented in this section, the dimension of the vectors $\mathbf{x}$, $\mathbf{y}$ and the matrix $\mathbf{A}$ corresponds to the number of degrees of freedom.

The novel data structure, the *BIN-CSR format*, is highly optimized for fast SpMV operations on GPUs. It uses a combination of a compressed sparse row storage (CSR) and a partitioning scheme grouping the matrix entries into bins. The data structure and arithmetic operations are implemented using CUDA [NVI15]. In general, SpMV operations for GPUs are memory bandwidth bound [BG09]. Therefore, it is very important to provide a GPU friendly memory access pattern in order to achieve an optimal throughput and high performance. The data structure is designed in such a way that each thread processes one row, i.e., each thread computes one value in the vector $\mathbf{y}$ for a SpMV operation. Similar to Oberhuber et al. [OSV11] the values are stored in a specific order, so that the threads can access the data coalesced. Therefore, the concept of a *bin* is introduced which is a portion of the matrix that a group of threads accesses concurrently (see Fig. 2.38 (b)). In order to achieve optimal performance, the width of such a bin should correspond to the number of threads that run concurrently on a multiprocessor. On Fermi architectures (compute capability 2.0) this is the size of a warp (32) or the size of a half warp (16) on older hardware (compute capability 1.X). In the examples and figures, a bin width of four is used for the sake of simplicity.

Fig. 2.38 (b) shows the conceptual layout of the BIN-CSR data structure. The diagonal is stored separately allowing an efficient Jacobi preconditioner. The matrix is compressed, i.e., only non-zero entries are stored with their corresponding column indices. These indices are stored in a separate array but in the same memory layout. The data structure therefore consists of four arrays: *diagonal*, *data*, *offset* and *col*. In contrast to existing CSR formats, the order of storage of the non-zero values is adapted, so that entries in the same compressed column are placed next to each other. Fig. 2.38 (c) shows the resulting off-diagonal matrix entries and column indices

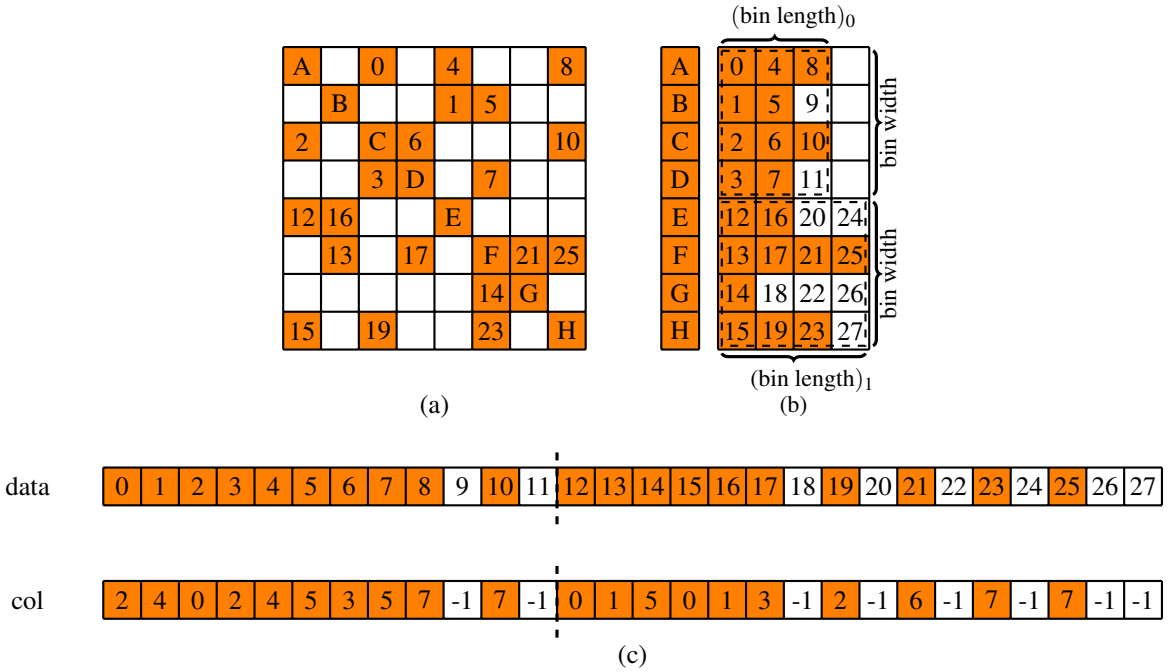


Figure 2.38.: (a) Sample of a sparse matrix. Colored squares represent non-zero entries. The numbers indicate the order in memory within the data structure and letters represent the diagonal. (b) compressed view of the sparse matrix with separated diagonal and off-diagonal part. Squares with numbers and a white background represent padded entries. Bin borders are indicated by dashed lines. (c) Value (*data*) and column index (*col*) arrays of the BIN-CSR data structure representing entries in the matrix and position in the respective row. Entries with padded values are indicated by white boxes, unused column indices are set to -1 and bin borders are indicated by dashed lines. In contrast to a standard CSR data structure, the data is reordered to achieve optimal memory throughput. In this simplified example, four consecutive entries can be processed in parallel.

in memory. Then, loading the matrix entries in a kernel can be done very efficiently: All threads in a warp can coalesce their memory request for each matrix column with only one memory fetch. In order to construct such a layout, all rows in each bin must have the same length. Therefore, the maximum row length per bin is determined:

$$\text{bin length}_i = \max_{j \in \text{bin}_i} \text{length}(\text{row}_j).$$

For each bin i, $(\text{bin length}_i) \times (\text{bin width})$ elements are allocated for data and column indices and the remaining data is padded to meet the memory alignment requirements. Furthermore, the offsets for each bin and row are computed and stored to provide access to the start of the non-zero entries for each row. The subsequent data can be obtained by incrementing the index by bin width elements. As in the CSR format, the data structure is then represented by the non-zero matrix data, the row offsets and the column indices.

A further extension of this data structure is motivated by the characteristics of the simulation applications. There, the global matrices exhibit the property that non-zero entries only arise in 3×3 blocks. Then, the number of memory reads for the column indices can be reduced to further speed up the operations. Note that in rigid

body simulation this assumption cannot be used for all constraints types. The extension, the *BIN-BCSR format* (BIN - block compressed sparse row), is conceptually similar to the work of Buatois et al. [BCL09] who also propose grouping non-zero entries in blocks. They determine clusters of 2×2 or 4×4 blocks of non-zero entries in a precomputation step to permit register blocking and maximizing the memory fetch bandwidth. Then, the processing of 2 or 4 rows, respectively, needs to be merged in one thread. But this considerably reduces the number of threads that are launched and may result in a low utilization of the multiprocessors. In contrast, 3×3 blocks are used here that only consist of non-zero entries and one thread is associated with each row. This keeps the number of threads high, resulting in higher occupancy. In Fig. 2.39 (a) an example of a sparse matrix with 3×3 block structure is shown. Fig. 2.39 (b) depicts a compressed representation of the off-diagonals and the separation of pure diagonal entries. In order to make use of the implicit representation of the column indices, the two off-diagonal entries of the diagonal 3×3 block in each row must be moved to the beginning of the data structure. Thereby, the size of column index array *col* can be reduced significantly by roughly a factor of 3 by storing only the first index in a row in each 3×3 block, as two out of three indices are implicit. Then, the column indices of the following two entries can be determined by incrementing the previous ones.

2.5.3.1. Sparse Matrix-Vector Multiplication

Using the proposed data structure results in a very fast SpMV operation $\mathbf{y} \leftarrow \mathbf{Ax}$, as the data of the matrix can be efficiently read from memory. Threads in a warp can load the non-zero values and the column indices coalesced. In contrast to other approaches (e.g., [BCL09,BB09a,MLA10]), this coalescing is achieved without a subsequent reduction step, as the data is ordered for one thread per row. Furthermore, as the size of each bin equals a multiple of the bin width, the offsets for each bin and for each row are naturally aligned. As the non-zero patterns of the matrices are highly irregular, texture memory is used to cache the vector $\mathbf{x}$ (similar to Bell et al. [BG09] and Baskaran et al. [BB09a]). Writing the results to the vector $\mathbf{y}$ is also coalesced as each thread processes one row.

```
1 def SpMV():
2     y[i] ← diagonal[i] ∗ x[i]
3     index ← offset[i];
4     endIndex ← offset[i + bin width];
5     while index < endIndex
6         y[i] ← y[i] + data[index] ∗ x[col[index]];
7         index ← index + bin width
```

Listing 2.7.: SpMV operation for row and thread i

Listing 2.7 shows the SpMV operation for the BIN-CSR matrix in pseudo code. The major differences compared to a CSR implementation are in line 1, 5 and 6. First, the result $y[i]$ is initialized by multiplication with the diagonal matrix element. Second, the index is incremented by the bin width. This results in high performance due to the minimal number of memory fetches. The off-diagonal access pattern for the SpMV operation is shown in Fig. 2.40. However, the data layout for optimizing coalesced loading leads to idle threads and increased memory consumption when the row lengths differ significantly within a bin. For the BIN-BCSR matrix data structure Listing 2.7 has to be slightly adapted: The multiplication in the while-loop must be performed three times for each column in a block. Furthermore, the column index must be read only once at the beginning of the loop and be incremented by one for the remaining two multiplications.

The proposed sparse matrix representation can be interpreted as a combination of the CSR and the ELLPACK (ELL) format [BG08,BG09]. The ELL format pads the data per row to the maximum row length and is therefore

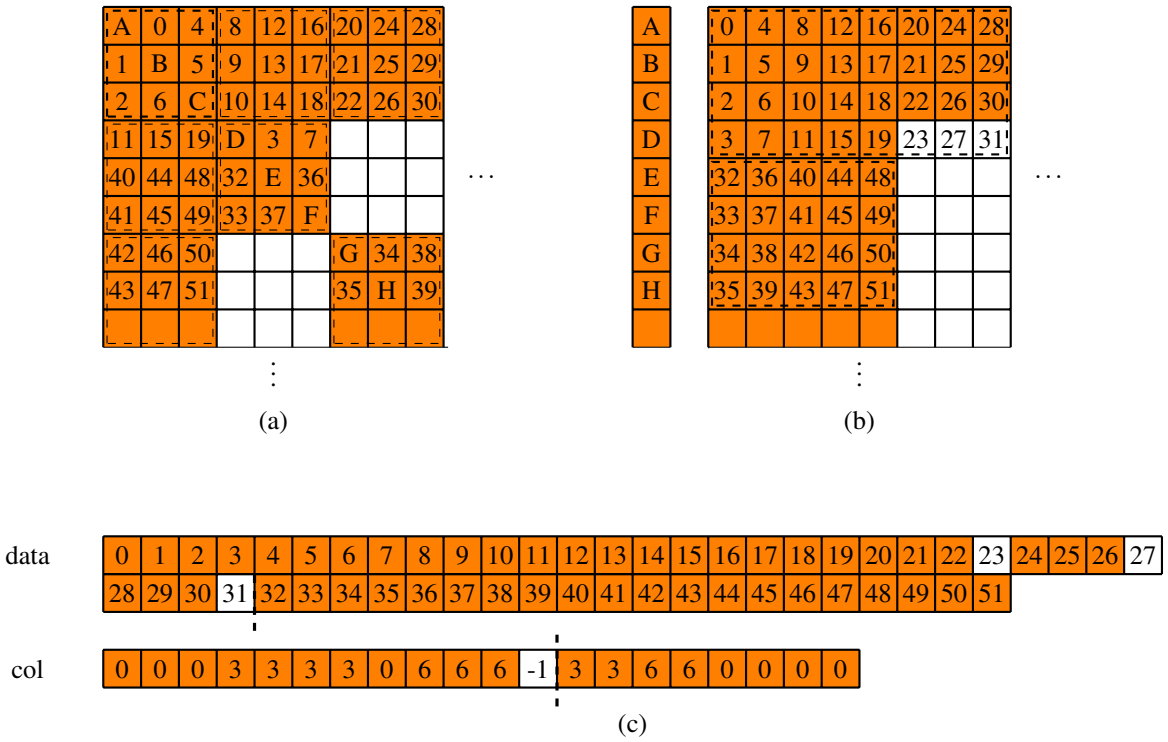


Figure 2.39.: (a) Excerpt of a sample sparse matrix with 3×3 block structure indicated by dashed boxes. Colored squares represent non-zero entries. The numbers indicate the order in memory within the data structure and letters represent the diagonal. (b) compressed representation of the sparse matrix with separated diagonal and off-diagonal part. The two off-diagonal entries of each diagonal 3×3 block are moved to the beginning of each row. (c) Value (*data*) and column index (*col*) arrays of the BIN-BCSR data structure representing entries in the matrix and position in the respective row. Bin borders are indicated with dashed lines. For consecutive entries within a 3×3 block, the column indices can be determined based on the previous entries enabling a more compact representation.

well suited for matrices with a nearly constant row length. As the efficiency of the ELL format degrades rapidly when the number of entries per row varies, Bell et al. [BG09] propose a hybrid (HYB) format combining ELL and coordinate (COO) data structure. It stores the majority of matrix entries in an ELL structure and exceptionally long rows in a COO format. However, it is complicated to determine a good criterion that adjusts the size of the ELL portion. Furthermore, the COO format suffers from the fact that its SpMV operation is based on a computationally expensive segmented reduction scheme and therefore multiple kernels are required for a single SpMV operation.

In the work of Vazquez et al. [VOFG10] an improved version of the ELL format, the ELLPACK-R format, is introduced. It uses an additional array to store the length of each row and transposes the data to achieve coalesced memory transactions. Thus, a very fast and efficient SpMV implementation can be achieved. However, in a scenario with only a few rows that contain large numbers of non-zero elements there is very high memory overhead for ELL-based matrix formats (e.g., for matrices of quadratic FEs, see Table 2.10). Monakov et al. [MLA10] propose a generalization with the Sliced ELLPACK format by grouping S adjacent rows and storing only row lengths

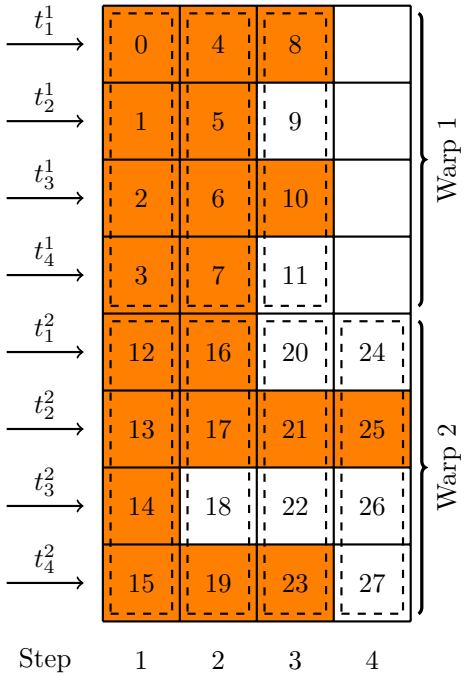


Figure 2.40.: The off-diagonal elements of the matrix in Fig. 2.38 (a) can be loaded in one coalesced memory transaction per step. Each memory request for the threads t_i^j $(i = 0,\ldots,3)$ in a warp j is indicated by dashed lines.

for these groups. Thus, the memory consumption is reduced significantly. As in the approach of Oberhuber et al. [OSV11], the BIN-BCSR data structure reorders the data and allows for varying bin lengths. This provides a good trade-off between memory and performance benefits. In the worst case, the memory consumption of the format is equal to ELL-based formats but in general it is significantly lower.

2.5.3.2. Kernel call minimization for PCG solver

When implementing a PCG algorithm, usually it is assembled out of a set of single operations like SpMV, dot products and AXPY, where the latter computes $\mathbf{y} \leftarrow \alpha\mathbf{x} + \mathbf{y}$. In the case of a CG algorithm, three AXPY operations, two dot products and one SpMV operation are needed in the inner loop [She94]. Using a PCG algorithm there is an additional matrix-vector multiplication with the preconditioner matrix. Here, this is computationally similar to a AXPY operation as a simple Jacobi preconditioner is used that only requires the multiplication with the inverted diagonal. Additionally, there are two filter operations that can be used for constraints (see Baraff et al. [BW98]). This allows for position and velocity control, e.g., to resolve collisions or to constrain the movement of nodes. Thus, implementing the PCG algorithm for CUDA results in up to nine subsequent kernel calls in the inner loop. As invoking such kernels naturally generates some overhead, it is desirable to minimize the number of calls. Particularly in interactive applications, the kernel call overhead can be in the same order of magnitude as simple kernels such as the AXPY operation. Moreover, the AXPY operation and similar kernels are heavily memory bandwidth bound. Merging operations into as few kernels as possible, can increase the number of arith-

metic operations per memory operation. Furthermore, it is desirable to avoid memory transfers between GPU and CPU, as they trigger a significant overhead.

```
1 i ← 0; r ← b − Ax; filter(r);
2 d ← M^{-1}r; δ ← r^T d; δ_0 ← δ;
3 while {i ≤ i_max and δ > ε^2 δ_0}
4     δ_old ← δ; q ← Ad; filter(q); α ← δ / (d^T q);
5     x ← x + αd; r ← r − αq; s ← M^{-1}r; δ ← r^T s;
6     β ← δ / δ_old; d ← r + βd; i ← i + 1;
```

Listing 2.8.: PCG algorithm

Listing 2.8 shows the PCG algorithm (cf. [She94, BCL09]) to solve the linear system

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \tag{2.127}$$

which may result from elastic or articulated body simulation (see Eq. (2.129) or Eq. (2.132)). Here, $\mathbf{M}$ is an appropriate preconditioner for $\mathbf{A}$ and $\mathbf{x}$ is initialized with a first guess. In the simulation applications presented in this section, the result from the last time step is used as a warm start for faster convergence. The *filter* operation allows for partially constraining values in the $\mathbf{x}$ vector that may be used for velocity correction (see [BW98] for details). In the novel merged conjugate gradient (MCG) approach, the initialization operations (in lines 1 and 2) as well as the operations in each line of the while-loop each are combined into one kernel. Merging the AXPY and SpMV-operations is simple, as the SpMV implementation can be executed per row. However, to incorporate the dot product, a special synchronization mechanism is needed since its result (e.g., α, β and δ) are required globally in all threads.

Generally, the proposed dot product computation is a reduction that consists of two phases: in the first kernel each block computes an intermediate result by partially reducing the associated data in shared memory. In the second kernel, each block loads all the intermediate data and again performs a reduction (see Fig. 2.41). So, the second reduction is computed redundantly by each block, but its result is then available in all blocks. However, for extremely large linear systems the number of required blocks can get high so that the second reduction slows down. But in these cases the performance gain of merging the kernels for solving the linear system would be low.

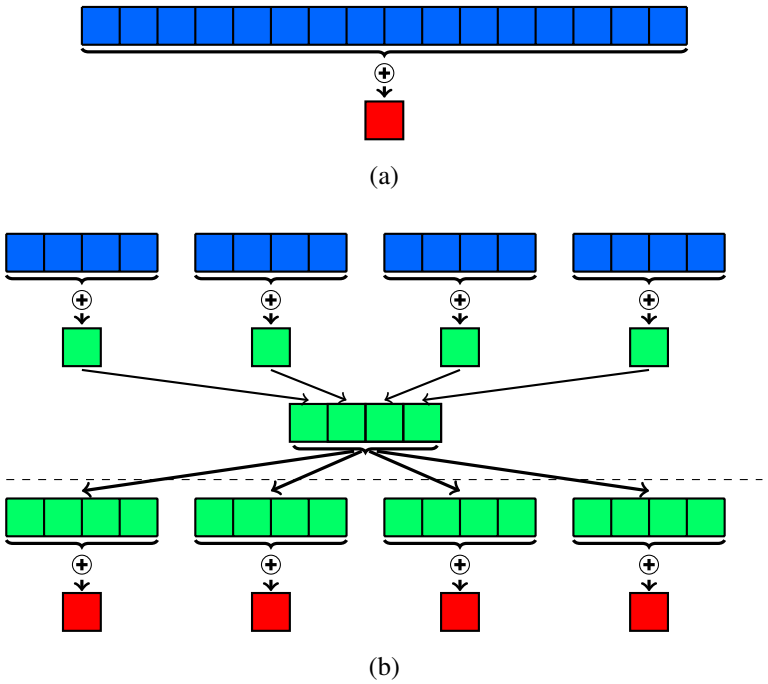
In order to minimize the overall number of kernel calls, the reduction is combined with the remaining computations (e.g. AXPY). As there are two dot products and the SpMV operation in the inner loop of each CG iteration that needs synchronization, the number of kernel calls can be reduced to three.

2.5.3.3. GPU matrix construction and update

An efficient matrix construction is crucial for applications, where the linear system changes in every iteration step. In the simulation applications, the update of every non-zero entry is computed by summation

$$A_{ij} = \sum_{k \in \Gamma_{ij}} A_k^e, \tag{2.128}$$

as a matrix entry is influenced by a few local element contributions A_k^e. The element entries A_k^e are stored linearly in memory. In a precomputation step, for every global entry A_{ij} the indices k referring to the positions in the



(a)

(b)

Figure 2.41.: (a) simplified example for a reduction of a 16 element input vector in blue and its result in red. (b) the parallel reduction is computed in four different blocks. The first kernel (above dashed line) partially reduces the input data (blue) and stores the result (green) it in global memory. As the final result is required in all blocks, the second reduction operates on the result of the first kernel (green) and redundantly computes the final result (red) in the second kernel (below dashed line).

element array are collected in the list of arrays Γ_{ij}. All the indices are stored linearly (*eIndices*) and an array (*eOffset*) is used for pointing to the start of a global entry. Listing 2.9 illustrates the assembly of the global matrix for one thread i that computes the i-th row. The access for writing the global matrix is the same as in the SpMV kernel and is therefore coalesced. The access pattern for reading the element matrix entries is irregular, so texture cache is used to speed it up.

In a typical application, where the global matrix changes in every iteration, the algorithm works as follows. First, all element matrix entries are computed on the GPU in parallel. As the updates of these element matrices are independent from each other, parallelization is trivial. One thread per element matrix entry A_k^e is used to recompute the necessary data. Afterwards, the global matrix is reassembled using Listing 2.9 with one thread per row. Finally, the system of linear equations is set up and solved on the GPU using Listing 2.8.

2.5.4. Simulation applications

In this section, two different simulation applications are described that serve as benchmarks for using the novel data structures. In Section 2.5.5, the usage of the BIN-BCSR data structure for these applications is evaluated. In both applications there is a system of linear equations that needs to be solved in every simulation step. The

```
1  def assembly():
2      index ← offset[i];
3      endIndex ← offset[i + bin width];
4      while index < endIndex
5          data[index] = 0
6          eStart ← eOffset[index];
7          eEnd ← eOffset[index + 1];
8          for l = eStart → eEnd − 1
9              k = [eIndices[l]];
10             data[index] ← data[index] + A^e_k;
11         index ← index + bin width
```

Listing 2.9.: Matrix assembly for row and thread i

corresponding matrices are large, symmetric and positive definite and have varying row sizes. Furthermore, the entries in the matrices change in every simulation step, whereas the matrix structures stay the same.

2.5.4.1. Elasticity simulation using FEM

For a realistic simulation of elastic bodies, several approaches have been presented that discretize the PDEs of elasticity using the FEM. Here, FEs with linear and quadratic basis functions are used, i.e. with 4 and 10 degrees of freedom per tetrahedron, respectively. There, a tetrahedral mesh discretization of the object is used and a corotational formulation is applied as described in Section 2.3. Applying the FE discretization results in a set of ODEs

$$\mathbf{M}\ddot{\mathbf{u}} + \mathbf{K}\mathbf{u} = \mathbf{f}^{\text{ext}}.$$

Here, $\mathbf{M}$ and $\mathbf{K}$ are the mass and stiffness matrices, respectively. The discretized displacement field $\mathbf{u} = \mathbf{X} - \mathbf{x}$ needs to be computed taking external and inertial forces $\mathbf{f}^{\text{ext}}$ and $\mathbf{M}\ddot{\mathbf{u}}$, respectively, into account. Using an implicit time integration scheme (see Baraff et al. [BW98]) results in a system of linear equations

$$\left(\mathbf{M} + \Delta t^2 \mathbf{K}\right) \Delta \mathbf{v} = \Delta t \left(\mathbf{f} + \Delta t \mathbf{K}\mathbf{v}\right), \tag{2.129}$$

where the resulting force $\mathbf{f}$ is computed by

$$\mathbf{f} = \mathbf{f}^{\text{ext}} + \mathbf{f}^{\text{el}} = \mathbf{f}^{\text{ext}} + \mathbf{K}\mathbf{u}. \tag{2.130}$$

Note that the time integration scheme introduces numerical viscosity, so damping forces are not considered explicitly. Applying corotation requires computing the rotation $\mathbf{R}$ for each element $\mathbb{T}$ by a polar decomposition (see [MG04]). The element stiffness matrices $\mathbf{K}_{\mathbb{T}} = \mathbf{R}^T\, {}^0\mathbf{K}_{\mathbb{T}}\mathbf{R}$ have to be updated in every time step with the initial stiffness matrices ${}^0\mathbf{K}_{\mathbb{T}}$. Therefore, the linear system in Eq. (2.129) with the global stiffness matrix $\mathbf{K}$ has to be reconstructed in every step. The system matrix $\mathbf{A} = \mathbf{M} + \Delta t^2 \mathbf{K}$ is sparse, i.e., there only are non-zero 3×3 blocks when two degrees of freedom share a tetrahedron. Using this formulation, it is additionally required to replace the elastic force $\mathbf{f}^{\text{el}}$ in Eq. (2.130) with the corotational force $\mathbf{f}_{\mathbb{T}}^{\text{corot}} = \mathbf{K}_{\mathbb{T}}\mathbf{x} - \mathbf{R}^T\mathbf{X}$ that must be summed up over all elements $\mathbb{T}$. By solving Eq. (2.129) the velocity changes $\Delta\mathbf{v}$ are obtained that are used to update the state of the tetrahedral mesh

$$\mathbf{x} = \mathbf{x} + \Delta t(\mathbf{v} + \Delta \mathbf{v}).$$

2.5.4.2. Simulation of articulated bodies

Articulated bodies introduce holonomic constraints of the form $\mathbf{C}(\mathbf{x},t) = \mathbf{0}$ to a system of rigid bodies in order to simulate joints. These constraints are simulated by using an impulse-based approach similar to [BS06] and [WTF06]. Therefore, all constraints are transformed into a general form $\mathbf{J}\mathbf{v} + \mathbf{c} = \mathbf{0}$ by differentiating the constraint function $\mathbf{C}$ w.r.t. time.

Analogous to the Lagrange multiplier method, the magnitudes λ of the impulses that are required to simulate a constraint could also be computed by solving

$$\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T\lambda = -\mathbf{J}\mathbf{M}^{-1}\mathbf{p}_{\text{ext}} - \mathbf{c}, \tag{2.131}$$

where the vector $\mathbf{p}_{\text{ext}}$ contains impulses and angular momenta which correspond to the external forces and torques in one time step. In this way, the impulses are computed in order to compensate the influence of external forces and torques. If this approach is used, an additional stabilization method is required to prevent the joints from breaking up [CP03].

Instead, a prediction of the constraint state is used in order to compute the required impulses which solve the stabilization problem [BS06]. The prediction $\mathbf{C}(\tilde{\mathbf{x}}, t + \Delta t)$ for a constraint is determined by solving the unconstrained equations of motion which gives a prediction $\tilde{\mathbf{x}}$ of the positions. Now, impulses are required that change the velocities of the bodies at time t so that the constraint is fulfilled after a time step of size Δt. This means that $\mathbf{C}(\mathbf{x}, t + \Delta t) = \mathbf{0}$ must hold.

To get the required impulses a non-linear equation must be solved since the relative motion of the bodies is generally non-linear. Weinstein et al. [WTF06] use Newton iteration to get the solution of this equation. In contrast to that, here the required velocity change is approximated by assuming a linear motion which results in $\Delta\mathbf{v} = 1/\Delta t \cdot \mathbf{C}(\tilde{\mathbf{x}}, t + \Delta t)$. Due to this linearization of the problem, the impulses can be efficiently computed by solving a system of linear equations. The required system

$$\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T\lambda = \Delta\mathbf{v} \tag{2.132}$$

is obtained by exchanging the right-hand side of the system of linear Eq. (2.131). As $\Delta\mathbf{v}$ is just an approximation, the resulting impulses will generally not eliminate the violation exactly. Therefore, the prediction and impulse computation is performed in an iterative process until a user-defined accuracy is reached. Note that in most cases the motion of the joint connectors is almost linear. Therefore, an average of just one or two iterations is sufficient to achieve an accuracy of 10^{-6} m in different test simulations with large external forces. For models with kinematic loops the matrix in Eq. (2.132) may become singular during the simulation. In this case, joints of the articulated body are removed in order to break up the problematic loops and additional impulses are added to mimic the effects of the kinematic loop as proposed in [BETC12].

For the simulation of a velocity constraint of the form $\mathbf{C}(\mathbf{v},t) = \mathbf{0}$, e.g., to perform a post-stabilization step [WTF06], the velocity difference $\Delta\mathbf{v}$ can be determined directly. Therefore, the linear system (Eq. (2.132)) must be solved just once in a simulation step to obtain an exact solution for the impulses. Note that the matrix of the system is constant for time t. Hence, it must be created only once per simulation step to compute the impulses of all holonomic and velocity constraints. The required matrix multiplication $\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T$ is performed on the GPU. Furthermore, the matrix is typically sparse as a block is non-zero if and only if two joints share a common body.

2.5.5. Results

In this section, the efficiency of the presented data structures is analyzed and compared to the performance of previous works. The tests were performed in single precision on an Intel Core 2 Quad Q6600 with 2.4 GHz

Matrix	Origin	Dim.	NNZ
Ship	CUSP	140,874	7,813,404
Bridge	RB sim.	41,550	448,668
Armadillo	Quadratic FE	46,812	3,620,142
Bunny	Quadratic FE	226,284	18,109,026
Pensatore	Linear FE	38,379	1,590,165
Cloth	Cloth sim.	76,800	3,865,140

Table 2.9.: Matrices used for the performance test. All matrices are quadratic and symmetric. Dimension represents the number of rows and columns, whereas NNZ specifies the number of non-zero entries. The cloth model consists of a rectangular 160×160 patch.

Matrix	Max.	Min.	Avg.	Overhead
Ship	99	21	52.5	2.4%
Bridge	42	6	7.8	3.6%
Armadillo	312	27	75.3	13.2%
Bunny	552	27	77.0	12.8%
Pensatore	69	15	38.4	10.8%
Cloth	48	18	47.3	0.4%

Table 2.10.: Maximum, minimum and average row lengths and memory overhead of the test matrices.

and a GeForce GTX 470 unless stated otherwise. CUDA SDK 4.0 and several matrices (see Table 2.9) that originate from the simulation applications and from the tests in the work of Bell et al. [BG09] were used. The matrix "Cloth" is obtained from a cloth simulation with a 160×160 patch (cf. Choi et al. [CK02]). Finally, it is demonstrated how different applications greatly benefit from these fast solving routines and discuss the properties of the proposed methods.

2.5.5.1. Performance Analysis

First, performance analysis of the SpMV routine that is the most time consuming part in the PCG algorithm is in the focus. Similar performance tests to work of Bell et al. [BG09] and using their CUSP library [BG10] are applied with texture memory cache enabled for the comparison. In this library, there are different kinds of matrix formats such as the hybrid (HYB), coordinate (COO), ELLPACK (ELL) and the (vector) compressed sparse row (CSR) format. The diagonal- (DIA) and the scalar-CSR format were omitted as both showed a rather low performance in the tests. The performance is evaluated in terms of GFlop/s, which is the number of (single-precision) floating point operations that are executed per second. In the case of a SpMV operation, this is twice the number of non-zero entries, as for each entry there is a multiplication with the corresponding value in the $\mathbf{x}$ vector and an addition to the $\mathbf{y}$ vector. For the tests, matrices taken from the simulation applications and the matrix "FEM/Ship" from Bell et al. (see Table 2.9) were used. Table 2.10 shows the varying row lengths in these matrices that are typical for irregular discretization schemes.

The charts in Fig. 2.42 show the performance of the CUSP variants and the proposed BIN-CSR and BIN-BCSR data structures for the test matrices. The measurements clearly show that utilizing the proposed data structures results in a significantly higher performance than the matrix types available in CUSP.

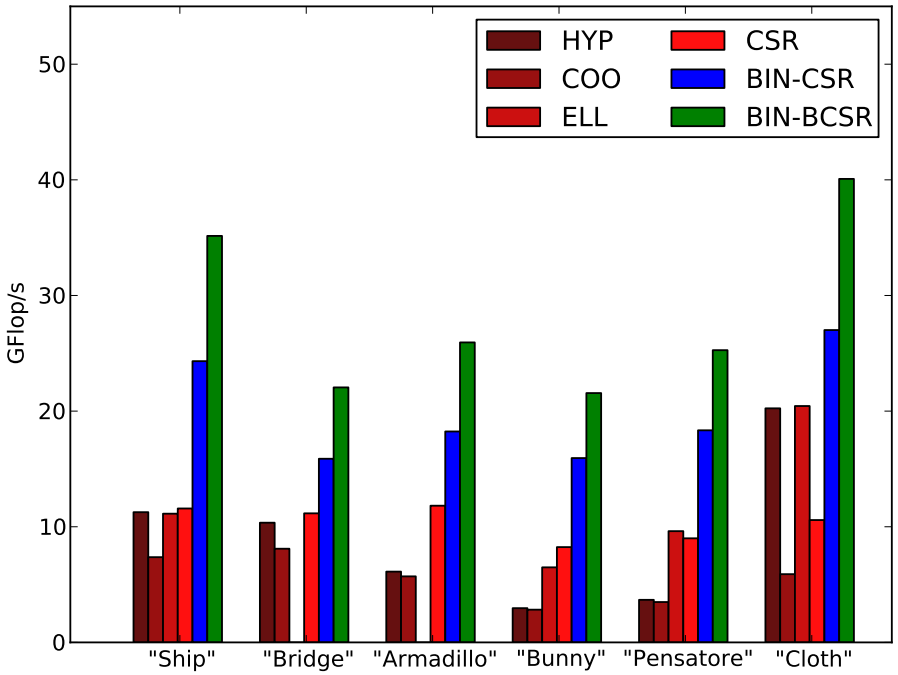


Figure 2.42.: Performance evaluation of SpMV operations with data structures available in the CUSP library (red colors) and the proposed BIN-CSR (blue) and BIN-BCSR (green) data structures on a GeForce GTX 470. Using the BIN-BCSR data structure results in performance increases by a factor of two to three in comparison to the best performing CUSP variant. The library failed for tests with the *Bridge* and the *Armadillo model* using the ELL format.

In order to evaluate the impact of the novel data structure and the MCG algorithm, several tests with matrices originating from simulations with linear FEs were performed. Therefore, 1000 iterations for each CG run were performed and the convergence criterion was removed to compare the methods. In Fig. 2.43, the curves show the speedup w.r.t. the number of non-zero entries of the matrices. Using the proposed BIN-BCSR data structure results in a nearly constant speedup by a factor of roughly three (black curve). There, a standard CG implementation with nine kernel calls has been used. Applying the proposed MCG algorithm results in an additional improvement (blue curve). This speedup is independent of the mesh resolution and the matrix size, since the saved overhead for kernel launches is constant. So, the curve and the impact of this optimization decrease with increasing resolution. However, this optimization is beneficial for meshes that are suitable for the interactive applications presented in this section.

Next, the convergence behavior of the MCG algorithm is evaluated and compared with a CPU implementation using single and double precision. Therefore, one system of linear equations from the *Armadillo scenario* (see Table 2.9) is solved. Fig. 2.44 shows the residuals w.r.t. the iterations. The convergence behavior of the GPU algorithm shows no significant difference compared to CPU single and double precision. The bottom diagram

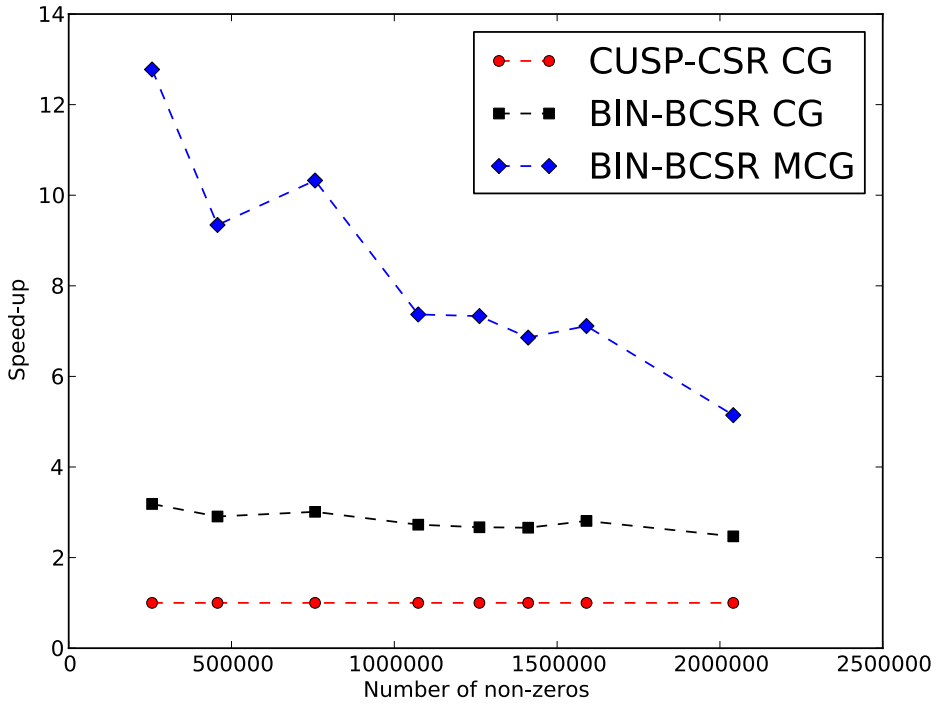


Figure 2.43.: Speedup of the BIN-BCSR data structure and the MCG algorithm w.r.t. the number of non-zero elements in the matrix: (Red) CUSP-CG algorithm. (Black) BIN-BCSR data structure with CG algorithm. (Blue) BIN-BCSR data structure with MCG algorithm.

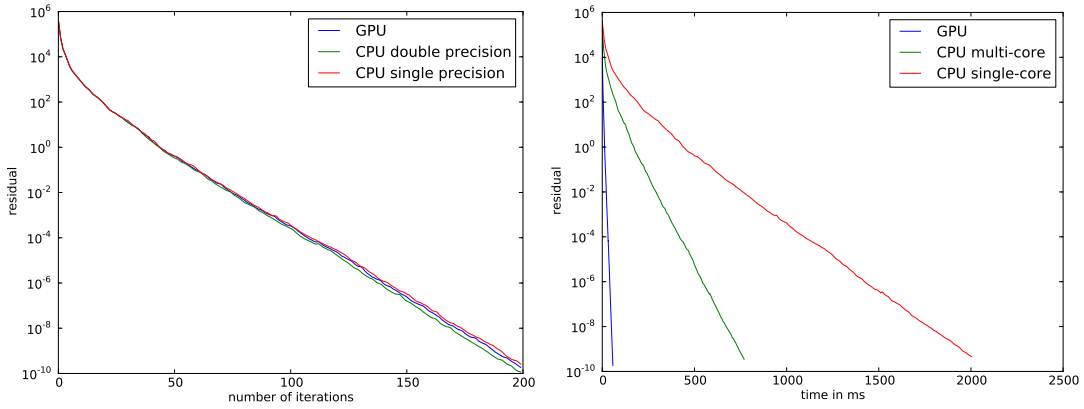
shows the residuals w.r.t. computation time of the GPU algorithm, of a multi-core and of a single-core implementation, all with single precision. For this test a GeForce GTX 580 and an Intel Core i7-2600 with 3.4 GHz were used. This graph clearly shows the significant performance increase using the proposed GPU solver.

2.5.5.2. Examples

In this section, it is shown how different simulations benefit from the proposed algorithms. Figure 2.35 shows snapshots of the simulations. The number of CG iterations was fixed to 30. Table 2.11 shows the timings for different simulation scenarios, where all CPU simulations were performed on a single core. These timings include the update of the element matrices, the assembly of the system matrix, force computation, the setup

Model	Simulation	CPU	GPU	Speedup
Bridge	Rigid Body	298	21	14.19
Armadillo	Quadr. FE	1009	75	13.45
Pensatore	Linear FE	713	33	21.61

Table 2.11.: Average times for one simulation step (in ms) with the CPU and GPU implementations, respectively. All CPU simulations run on one core only.



(a) (b)

Figure 2.44.: Convergence behavior of the CG algorithm solving a linear system arising from quadratic FEs. (a) Convergence behavior for GPU, CPU single precision and CPU double precision w.r.t. the number of iterations. (b) Convergence behavior w.r.t. computation time with GPU, multi-core and single-core implementation with single precision.

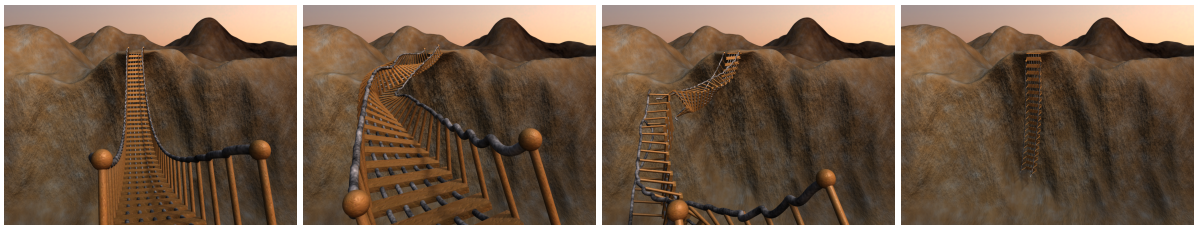


Figure 2.45.: Simulation of the *Bridge model* with rigid body elements.

and solution of the linear system. With a multi-core CPU implementation a speedup of approximately three was achieved on the four available cores. In all cases, an interactive simulation on the CPU with the simulation models is not realizable as the time for computing one simulation step is far too high. However, with the proposed GPU-based solver, interactive frame rates are achieved allowing for more degrees of freedom implying higher accuracy. A key factor of the speedup is that there is no memory transfer between CPU and GPU memory during simulation.

Up to 100K tetrahedral linear FEs with 30K degrees of freedom can be simulated at interactive rates. For quadratic FEs with ten nodes per tetrahedron interactivity can be achieved with up to 13K tetrahedral elements with 28K degrees of freedom. In both cases, the assembly of the system matrix takes only a fraction of the whole simulation step. Examples are shown in Fig. 2.35. Fig. 2.45 shows the interactive simulation of a complex *Bridge model* which consists of 14K rigid bodies and has a system of linear equations with a dimension of 41K. Interactive cloth simulations can be performed with more than 26K particles.

2.5.5.3. Discussion

The proposed data structures are designed for fast SpMV operations. This is achieved by optimally coalesced memory loads without the need for a subsequent reduction of any intermediate result. It is very well suited for large sparse matrices with varying row sizes that typically arise in FE discretizations or in other methods with local dependencies. Table 2.10 shows the high variations in length of rows among the test matrices. The adaption of the row lengths per bin provides a good trade-off between memory overhead and performance benefits. The current implementation uses single precision. In all of the applications no problems were noticed w.r.t. convergence or accuracy. The data structure is not suited for dense matrices, as there is a memory and performance overhead due to the additional information of offsets and column indices. The additional memory consumption for the padding is rather low, as it depends locally on the row length within a bin (see Table 2.10). E.g., in the ELLPACK-R format [VOFG10] the data is padded depending on the maximum row length of the whole matrix. This format is more suitable for matrices with nearly constant row sizes.

There are other approaches optimizing the access pattern for coalesced loading on the GPU. The HYP-format [BG09] and the optimizations by Baskaran et al. [BB09a] use one thread per matrix entry. To compute the final result a subsequent reduction is needed, which either requires an additional kernel call.

Buatois et al. [BCL09] presented a method where non-zero entries are clustered in 2×2 and 4×4 blocks. This clustering enables the reuse of values in the y-vector, but this can also be achieved with the texture memory cache. Furthermore, they use a compressed row storage format (CRS), a synonym for CSR, that does not take memory alignment into consideration. So, they do not exploit coalesced loading reducing the effective memory bandwidth, which is crucial for high performance.

Allard et al. [ACF11] presented a linear FE simulation running completely on the GPU. A major difference to this work is the emulation of SpMV by computing the individual element contributions separately. This emulation is comparable to the proposed GPU matrix construction step which is very difficult to optimize as the memory access patterns are highly irregular. It is beneficial to perform this operation only once per simulation step. It also important to note that for a larger number of CG iterations the proposed method is more efficient since a faster SpMV operation has a higher impact on the overall performance.

In the simulation of articulated bodies with kinematic loops the matrix of the system of linear equations which has to be solved may become singular and a special treatment is required. However, the *Bridge model* (see Fig. 2.45) contains several closed loops and the matrix never got singular during the whole simulation.

For the matrix update, the dependencies between the local and global matrix entries must be determined in a precomputation step. If a method requires topological changes of the mesh, these dependencies must be recomputed. In this case, the proposed matrix update is not suitable without modification.

2.5.6. Summary

In this section, novel data structures and methods for massively parallel computation of deformation simulation have been presented. Due to a special memory layout and an implicit modeling of column indices for a 3×3 block structure, the *BIN-BCSR data structure* enables fast SpMV. Thereby, dynamic simulations in physically based simulation applications can be accelerated significantly. In addition, the *MCG method* has been developed, which reduces the number of kernel calls and is therefore especially beneficial for interactive applications. In order to allow for fast simulations of corotational elasticity, an efficient assembly method for constructing the matrix in *BIN-BCSR* format has been presented. Applying these data structures and methods significantly accelerates simulation of dynamic volumetric deformation.

The next section summarizes the contributions of this thesis for simulation of dynamic volumetric deformation. The benefits of the novel approaches are stated and the methods are discussed regarding limitations and insights gained during the investigations.

2.6. Contributions and conclusions

In this section, the contributions made to simulation of dynamic volumetric deformation are summarized. Additionally, the findings and limitations of the novel approaches are discussed.

2.6.1. Contributions

In this chapter, the following novel methods improving the quality and efficiency of simulations of dynamic volumetric deformation have been presented:

1. The *quadratic BB-form method* represents the field of displacements in a corotational approach more accurately.
2. In order to further accelerate the computation in each time step, the *mesh-based matrix assembly* for higher-order FEs provides fast traversal for matrix updates. These updates are necessary when applying corotational elasticity.
3. The higher-order FE approach has been extended to the *cubic BB-form method*, which further enhances the accuracy and fidelity for deformation simulation.
4. The *p-multigrid method* makes use of a polynomial hierarchy in order to efficiently solve the system of equations for higher-order discretizations. This method can be directly applied as an iterative method or as a preconditioner for a CG algorithm.
5. The *BIN-BCSR data structure* for sparse matrices is specialized for massively parallel computing. The memory layout is designed such that memory accesses are contiguous and result in near optimal usage of memory bandwidth for GPUs.
6. The *MCG method* reduces the number of required kernel calls to solve the linear system. The associated overhead due to kernel execution is minimized and results in an additional speedup.

The proposed methods significantly accelerate the simulation of physically based deformation simulation. Table 2.12 summarizes the contributions with their individual improvements.

The adoption of quadratic FEs increases the accuracy of simulations and reduces computation time. By increasing the individual elements' modeling power in comparison to linear FEs, higher-order polynomials can represent qualitatively comparable deformation with significantly fewer elements. In addition, higher-order FEs are better suited to simulate near-incompressible material in contrast to standard linear FEs, as shear locking is avoided due to the additional degrees of freedom per element. Furthermore, applying a higher-order discretization reduces the number of required polar decompositions in a corotational approach because of fewer elements are needed. In addition, applying BB-form FEs results in a more concise approach as an analytic computation of stiffness matrix entries is possible. This additionally allows for a straightforward approximation of rotations for corotational elasticity, as the integrals do not have to be evaluated at predefined integration points. This results in an acceleration of rotation field extraction by a factor of four for quadratic and ten for cubic FEs. When utilizing an efficient solver or applying only a limited number of iterations for the solving the linear system, the setup for the linear systems incorporating the new stiffness matrix becomes a bottleneck. Therefore, the *mesh-based matrix assembly* was developed, resulting in an acceleration by approximately 75% in comparison to a standard approach. In summary, the novel approaches accelerate volumetric deformation simulations by a factor of three in comparison to previous quadratic FE methods. It has been discovered that simulations with higher-order bases significantly improve the fidelity of animations with a comparable computational effort to linear FEs. In general, the acceleration of using quadratic instead of linear FEs is difficult to quantify, as two equivalently accurate discretizations cannot be defined. However, for a specific example an acceleration of a factor of five is

Method	Publication and section	Main result
Quadratic BB-form method	[WKS*11], Section 2.3	Acceleration by a factor of three compared to [MTPS08]. Significantly fewer elements are required compared to linear FEs.
Mesh-based matrix assembly	[WKS*11], Section 2.3	75% faster construction of linear systems.
Cubic BB-form method	[WMRA*14], [WMRA*15], Section 2.4	Significantly higher accuracy compared to quadratic and linear FEs.
p-multigrid method	[WMRA*14], [WMRA*15], Section 2.4	Acceleration of the solution of linear systems by up to a factor of seven in comparison to a CG algorithm.
Merged conjugate gradient method	[WBS*13], Section 2.5	Acceleration of a CG algorithm on the GPU by a factor of three to four for interactive scenarios.
BIN-BCSR matrix data structure	[WBS*13], Section 2.5	Acceleration of SpMV by a factor of two to three in comparison to state-of-the-art GPU data structures.

Table 2.12.: This table shows the contributions of this chapter and summarizes the corresponding results. Furthermore, it provides a reference to the section where it is described and to the corresponding publication.

achieved. Nevertheless, complex deformations can be simulated with a significantly lower number of elements and therefore less computation time.

The extension to cubic BB-form FEs further increases the accuracy of simulations. It has been demonstrated that a simulation with cubic FEs exhibits superior convergence (see Table) meaning that only a small number of elements are required to achieve accurate results. Furthermore, this kind of modeling enables the construction of efficient multigrid solvers based on discretizations with varying polynomial degree. The *p-multigrid method* is based on building a BB-form polynomial hierarchy and accelerates the linear solve of up to a factor of seven. Especially, when artificial viscosity due to the non-accurate solution of the linear system is disturbing, a *p-multigrid approach* is beneficial as the convergence behavior is more predictive as, e.g., a CG algorithm.

Exploiting the high degree of parallelism of GPUs has an additional potential of accelerating simulations. Especially the solution of the linear system can benefit from massively parallel computing. Although standard implementations of GPU data structures can be used, the adoption of the *BIN-BCSR data structure* for sparse matrices results in an additional speedup. Compared to the state of the art, SpMV is accelerated by a factor of two to three resulting in a significant speedup for the simulations. In addition, the *MCG method* was developed to minimize the number of kernel calls on the GPU when solving the linear system. The overhead is mitigated by executing only three instead of nine kernels within the main loop of a CG algorithm. This especially pays off for moderately sized discretizations and interactive scenarios where the constant overhead has a significant influence on the overall computing time. In summary, the novel approaches result in an acceleration by a factor in the order of 20 in comparison to single-core CPU implementations.

2.6.2. Discussion

The novel methods presented in this chapter provide a significant acceleration for physically based volumetric deformation simulation. The goal of increasing accuracy in simulations while reducing computation time has been achieved. The limit of mesh resolution for interactivity has been extended. With a GPU-based approach, up to 100K elements for linear and 13K elements for quadratic basis functions can be used to compute the dynamics of volumetric deformation in interactive time, i.e., with ten or more updates per second.

The representation of deformation with higher-order bases has several advantages. As described above, complex deformation can be represented more accurately. Especially, torsional deformation can be recreated more precisely than with linear elements. Furthermore, the analytic integration, multiplication and differentiation of BB-form FEs ease the implementation of physically based simulation. This is due to the fact that the basis functions are defined w.r.t. barycentric coordinates, which are independent from coordinate systems by definition. In contrast, standard FEs require transformation back to a reference element and evaluation w.r.t. degree-dependent integration points. In addition, the analytic evaluation of integrals allows for an easy method for rotation approximation. Furthermore, the reduction of shear locking effects, as observed by Mezger et al. [MTPS08] for standard quadratic FEs, is also reproduced with higher-order BB-form FEs. This allows for the simulation of near-incompressible materials, as the additional degrees of freedom on edges can compensate for the volume change when pure shear is present.

The computation time of corotational elasticity with quadratic elements is affected by various factors. The number of elements determines the number of polar decompositions that have to be conducted. The number of degrees of freedom (three per node) determines the size of the linear system. In a FE discretization the sparsity pattern depends on the topology of the tetrahedral mesh and the order of the bases. All pairs of nodes that share a tetrahedron correspond to a non-zero entry in the matrix. Therefore, every edge corresponds to a non-zero entry in the matrix for linear FEs. For quadratic and cubic FEs, the non-zero entries are given by the nodes that share are tetrahedron, which is more complex to determine as nodes are also associated with edges and

faces. The number of non-zeros is important for the estimation of the cost of SpMV or Gauss-Seidel iterations. Furthermore, the adjacency relations between elements determine the number of element matrix entries that have to be summed up to construct the sparse matrix. Therefore, the time complexity for simulating corotational elasticity with higher-order FEs is a combination of vertex, edge, face, element counts and the connectivity of the tetrahedral mesh. When using a multigrid algorithm to compute the solution of the linear system, all of the building blocks have a linear time complexity or better w.r.t. their corresponding quantity (vertices, elements, etc.).

Solving the linear system is usually one the of most computationally intensive parts of corotational simulation algorithms. However, when utilizing an efficient solver or limiting the iterations it was found that the linear system is no longer the only bottleneck. The setup of the linear system, which must be conducted in every simulation step is then also a large contributor to the computation time. Ideally, the linear system is constant and the matrix can be prefactored only once or a constant multigrid hierarchy can be evaluated in a precomputation step. However, the corotational approach requires an update of the element matrices and the reconstruction of the linear system. For future work it would be interesting to come up with a formulation that improves the corotational method such that a constant stiffness matrix is at the core of the system. This might be achieved by defining degrees of freedom that measure the deformation locally relative to the discretization.

Cubic FEs are a further extension based on increasing the polynomial degree resulting in higher accuracy. However, in its current formulation higher-order modeling has the drawback that only rest shapes with flat triangles can be represented. In order to improve the geometry approximation with flat surface triangles, many tetrahedral elements are required, especially for geometry with highly varying curvature. Resolution of all small scale details is difficult to achieve with a small number of elements. It might be worth investigating how curved surfaces can be incorporated into the BB-form setting. However, the property of analytical integration will vanish as the transformation from a flat reference element to the curved rest shape must be considered. Nevertheless, the coupling approach described in Section 2.3 provides a reasonable method for simulating with a coarse mesh and using a fine triangular mesh for visualization.

The *p-multigrid method* results in a significant acceleration for solving the linear system when using higher-order discretizations. However, as the algorithm especially pays off for large models, the resulting scheme is not yet interactive with the evaluated models. Although the computation times are in the order of seconds, this is not sufficient for interactivity, which requires a frame rate of 10Hz. A massively parallel version of the *p-multigrid method* would be an interesting option for future work to finally achieve interactivity for large model sizes. However, in this case, an alternative smoothing operation must be used, as the standard Gauss-Seidel algorithm is not amenable to parallelization. An option would be to use a mesh coloring approach, which splits the nodes into independent sets. Then smoothing could be executed in parallel for one "color" in order to avoid race conditions. An interesting question then is whether the increased number of kernel executions still pays off or if a Jacobi smoother is a better choice despite its weaker convergence.

The adoption of the specialized *BIN-BCSR data structure* for GPUs decreases computation time as SpMV operations are significantly accelerated. Thereby, linear systems can be solved more quickly with a CG algorithm, as SpMV is the most time consuming operation in this approach. Although the reorganization of data and resulting coalesced memory access are also beneficial for a standard CSR representation, a considerable benefit is to exploit the 3×3 structure of FE discretization for volumetric deformation. There, the column indices can be implicitly reused within the 3×3 blocks resulting in fewer load operations.

The proposed GPU method uses single precision. For general simulation applications it might be necessary to switch to double precision for floating point numbers for numerical reasons. For a possible extension to double precision, the data layout of the *BIN-BCSR data structure* must be revised in order to achieve optimal coalescing. Furthermore, the performance benefit might be reduced, as GPUs are less efficient when using double precision.

It was found that the overhead for launching kernels is significant. This is especially valid for moderately sized discretizations, where the constant overhead significantly influences the runtime. Therefore, the merging of the kernels is especially beneficial for interactive simulations. For the analyzed data sets this resulted in accelerations by a factor of more than three.

The general approach of utilizing parallel computing and distributing computations among several processors is a viable option. Especially GPUs with thousands of cores seem to be a good option for minimizing computation time. The acceleration factors for simulating volumetric deformation are on the order of up to 20. This factor seems to be quite high. However, the acceleration is estimated w.r.t. a single core implementation. It would be interesting to compare this to a multi-core implementation. It might turn out that the acceleration is only moderate. However, in interactive simulation scenarios the simulation results are directly visualized. Therefore, the data transfer is avoided by directly interpreting the simulation results as vertex positions for rendering.

3. Fluid simulation

Realistic fluid dynamics are an important ingredient for computer generated animation, interactive virtual reality or virtual environments. Many of the statements and requirements from the previous chapter also apply to the simulation of fluid dynamics. Physically based modeling can generate visually realistic motion, as the partial differential equations (PDEs) are able to capture all characteristics of a fluid. Although there are geometrically motivated methods which can mimic the motion, employing continuum mechanical modeling based on conservation laws will presumably result in more realism. In the case of fluid dynamics, the governing equations are the *Navier-Stokes* or *Euler equations* which are used in this thesis as a basis for the computations. As for volumetric deformation, the PDEs of fluid dynamics need to be discretized w.r.t. time and space. In computer animation, time discretization is usually conducted by separating the operators of the PDE and applying a simple time integration scheme to each of them. Regular grids with uniform spacing are a common choice for discretizing space and finite difference (FD) or finite volume method (FVM) are frequently adopted. These grids have the advantageous property of regular access patterns that implicitly result in cache coherency. Similar to volumetric deformation, sufficiently small element sizes will result in accurate solutions and varying this mesh resolution allows for a trade-off between computation time and accuracy. With standard methods, the resulting resolution will be comparatively low to achieve interactive computation times, which in turn might not resolve all desired detail.

The objective of this thesis is to develop fast methods for physically based simulation in order to increase resolution and achieve interactive rates at the same time. In this chapter the developed methods are applied to the domain of fluid simulation. For a given time budget, faster methods allow to increase the detail of motion and therefore realism. This goal is achieved by developing new contributions that can be associated with the three pillars: *more accurate discrete representations*, *efficient methods for linear systems*, and *data structures and methods for massively parallel computing*. These are (also shown in Fig. 3.1 with classification):

- In order to increase the accuracy on regular grids, the *geometric cut-cell method* is introduced. This method models sub-grid features for solid-fluid boundaries by explicitly computing geometric intersections. This discretization preserves the computationally attractive access patterns of regular grids and results in accuracy that could otherwise only be achieved by adaptivity with other mesh types, e.g., tetrahedral meshes.
- In order to accelerate the solution of linear systems, the *cut-cell geometric multigrid method* is developed, significantly accelerating computations in comparison to existing methods. Based on the *geometric cut-cell method*, compatible representations of the discretizations are generated on all levels of the multigrid hierarchy, which significantly increases the convergence rate. The algorithm is designed in such a way that all buildings blocks are parallelizable to be efficiently executed on graphics processing units (GPUs) and multi-core CPU architectures. Furthermore, multigrid compatible *symmetric Dirichlet boundary conditions* for FVM discretization are derived in order to simulate free-surface flows on this type of discretization with symmetric linear systems.
- Additionally, a GPU-based fluid simulation method is developed that parallelizes the computational building blocks and significantly accelerates the calculations. As many GPU algorithms are memory bandwidth bound, an *implicit matrix data structure* is developed that reduces the amount of required data transfers for a regular grid discretization and therefore accelerates the computations.

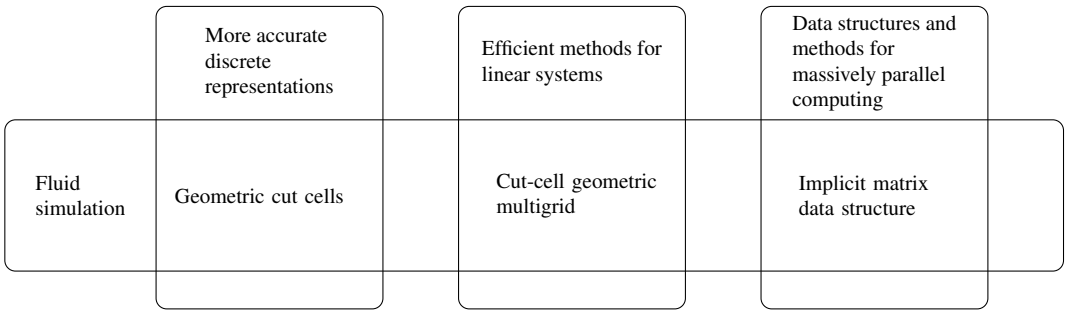


Figure 3.1.: The contributions of this thesis for fluid simulation are associated with the three pillars *more accurate discrete representations*, *efficient methods for linear systems*, and *data structures and methods for massively parallel computing*.

The *geometric cut-cell method* is an alternative discrete representation of simulation geometry on regular grids. Face fractions based on fluid-solid coverage are determined and incorporated into the equations to provide sub-grid accuracy for fluid simulations. The GPU method for fluid simulation uses the cut-cell discretization and provides a massively parallel approach to fluid simulation. Furthermore, employing the specialized *implicit matrix data structure* results in an additional speedup of 20% further accelerating the expensive matrix operations in comparison to a standard representation. In summery, the GPU approach accelerates the computations up to a factor of 14.

The *cut-cell geometric multigrid method* builds on top of this discrete representation and generates consistent discretizations on all levels of resolution. This efficient method significantly accelerates the solution of linear equations, which are the most time-consuming part in fluid simulation. It is significantly faster in comparison to standard numerical algorithms and results in a speedup of up to a factor of 3 in comparison to existing multigrid methods.

This chapter is organized as follows: First, the continuum mechanical modeling of fluids and its numerical discretization are described Section 3.1. There, the background regarding PDEs and numerical techniques for fluid simulation is summarized. Furthermore, this chapter defines the terminology for fluid simulation that is required for reviewing related work and introducing the new methods. Then, in Section 3.2 the related work in the context of fluid simulation is reviewed. In Section 3.3, a GPU-based fluid simulation method and the *implicit matrix data structure* are presented. Furthermore, the *geometric cut-cell method* is introduced that improves accuracy in fluid simulations on regular grids. Section 3.4 covers the *cut-cell geometric multigrid method*, where a significant acceleration for the solution of linear systems is achieved. Section 3.5 concludes this chapter and summarizes the findings and results.

3.1. Background

This section covers the mathematical and physical theory for simulation of fluid dynamics. Furthermore, the relevant terminology for computer animation is defined. Starting from the equations of motion for elastic solids derived in Section 2.1, the PDEs for incompressible fluids are derived in Subsection 3.1.1. Subsection 3.1.2 describes basic numerical schemes that discretize the equations w.r.t. space and time in order to compute the fluids' motion. The standard numerical method is outlined, which serves as reference for the approaches developed in this chapter.

3.1.1. Continuum mechanics

In this subsection, the continuum mechanical formulation for fluids is derived. A broader introduction to the modeling of fluid dynamics is given in the standard text book by Spurk and Aksel [SA07b]. The PDEs that are required for this chapter are derived in the following.

Fluid is the generic term for both gases and liquids. In contrast to elastic solids, fluids do not resist to forces that arise due to shear deformation. They deform constantly when exposed to such forces and initially nearby fluid particles will usually separate. This implies that fluids will not return to their initial state. However, shear rates generate viscous forces.

In the previous chapter the conservation of momentum for elasticity was derived in Eq. (2.35), which is a good point of departure for modeling fluids. Given a *simulation domain* Ω with its boundary Γ, this equation relates the change of momentum to elastic and external forces per unit volume. However, this equation must be adapted to account for the characteristics of fluids as purely elastic forces are not present. In fluid dynamics, the equations are stated w.r.t. velocity instead of displacement fields that are used in elasticity. Therefore, the acceleration term in Eq. (2.35) is replaced with the time derivative of velocity

$$\rho \frac{\mathrm{d}\mathbf{v}}{\mathrm{d}t} = \nabla \cdot \sigma + \rho \mathbf{b}. \tag{3.1}$$

This equation is formulated from a *Lagrangian viewpoint*. In this approach, the time evolution of the material, which is attached to the moving computational domain Ω, is observed. Therefore, the velocity $\mathbf{v}$ in Eq. 3.1 is denoted w.r.t. the initial coordinate system $\mathbf{x}$. The change of velocity in the Lagrangian perspective is evaluated by simply applying the partial derivative w.r.t. time $\frac{\mathrm{d}\mathbf{v}(\mathbf{x})}{\mathrm{d}t} = \frac{\partial \mathbf{v}(\mathbf{x})}{\partial t}$. In contrast to elasticity, fluids are usually described from an *Eulerian viewpoint*. In the Eulerian approach, the physical quantities are observed from fixed points in space and usually the domain Ω does not change. Therefore, the movement of material within the domain has to be considered when evaluating the velocity $\mathbf{v}(\mathbf{X})$. The difference between the Lagrangian and Eulerian viewpoint is depicted in Fig. 3.2. In order to account for the movement, the total derivative of $\frac{\mathrm{d}\mathbf{v}(\mathbf{X})}{\mathrm{d}t}$ must consider the mapping function $\mathbf{X} = \varphi(\mathbf{x},t)$ written as $\mathbf{X}(\mathbf{x})$ and derived in the previous chapter (see Eq. (2.2)). With this mapping, the velocity can be expressed in terms of the fixed domain $\mathbf{v}(\mathbf{X}) = \mathbf{v}(\varphi(\mathbf{x},t),t) = \mathbf{v}(\mathbf{X}(\mathbf{x})t)$. When evaluating the total derivative, the chain rule is applied that results in

$$\frac{\mathrm{d}\mathbf{v}(\mathbf{X},t)}{\mathrm{d}t} = \frac{\mathrm{d}\mathbf{v}(\mathbf{X}(\mathbf{x}),t)}{\mathrm{d}t} = \frac{\partial \mathbf{v}(\mathbf{X}(\mathbf{x}),t)}{\partial t} + \frac{\partial \mathbf{v}(\mathbf{X}(\mathbf{x}),t)}{\partial \mathbf{X}} \frac{\mathrm{d}\mathbf{X}}{\mathrm{d}t} = \frac{\partial \mathbf{v}(\mathbf{X}(\mathbf{x}),t)}{\partial t} + \frac{\partial \mathbf{v}(\mathbf{X}(\mathbf{x}),t)}{\partial \mathbf{X}} \mathbf{v}(\mathbf{X}) \tag{3.2}$$

or in short

$$\frac{\mathrm{d}\mathbf{v}}{\mathrm{d}t} = \frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v}. \tag{3.3}$$

This relation is called the *material derivative*, where additionally the *advection term* $\mathbf{v} \cdot \nabla \mathbf{v}$ appears in the Eulerian perspective. This reflects the fact that the change of velocity is influenced by the velocity itself. Restating

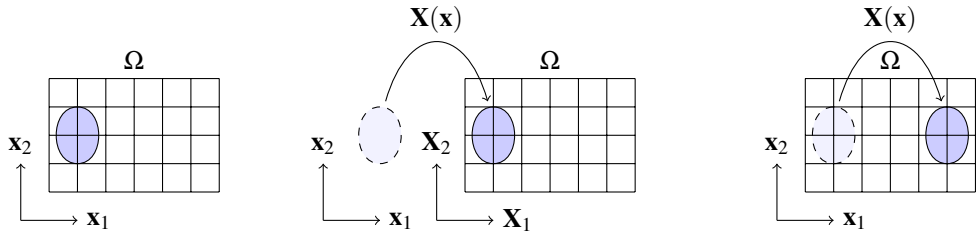


Figure 3.2.: An example showing the difference between Lagrangian and Eulerian viewpoints similar to Fig. 4 in the work of Fan et al. [FLLP13]. (a) Rest configuration with initial coordinate frame **x**. (b) In the Lagrangian perspective the computational domain moves with the simulated geometry. (c) In the Eulerian perspective the geometry is observed from fixed points in space. Therefore, the relative movement within the domain Ω given by the mapping $\mathbf{X}(\mathbf{x})$ has to be taken into account when deriving the equations of motion.

Eq. (3.1) by inserting the material derivative of Eq. (3.3) results in

$$\rho\left(\frac{\partial \mathbf{v}}{\partial t}+\mathbf{v}\cdot\nabla\mathbf{v}\right)=\nabla\cdot\sigma+\rho\mathbf{b}, \tag{3.4}$$

which accounts for the Eulerian perspective.

For fluid dynamics, the *Cauchy stress tensor* is split into two parts

$$\sigma=-\nabla p+\tau \tag{3.5}$$

with *pressure* p and the *deviatoric stress tensor* or *viscous stress tensor* τ. The deviatoric stress tensor τ represents force densities that arise due to friction because of velocity differences. It is symmetric and traceless, meaning the sum of the diagonal entries is zero. The pressure can be determined from σ by evaluating the *mean normal stress*

$$p=-\frac{1}{3}(\sigma_{xx}+\sigma_{yy}+\sigma_{zz}). \tag{3.6}$$

Inserting the separation of pressure and viscous stress tensor (Eq. (3.5)) into Eq. (3.4) results in

$$\rho\left(\frac{\partial \mathbf{v}}{\partial t}+\mathbf{v}\cdot\nabla\mathbf{v}\right)=-\nabla p+\nabla\cdot\tau+\rho\mathbf{b}. \tag{3.7}$$

In order to determine the entries of the viscous stress tensor τ, a further quantity is required, the so-called *strain rate*. This quantity is the time derivative of the strain tensor, which was derived in Section 2.1 in the context of deformation simulation. More specifically, it is the time derivative of the linearized Cauchy strain given in Eq. (2.16)

$$\dot{\varepsilon}=\frac{\partial \varepsilon}{\partial t}=\frac{1}{2}\left(\nabla\mathbf{v}^T+\nabla\mathbf{v}\right). \tag{3.8}$$

This pointwise quantity measures the change of deformation over time, i.e., the difference of velocity in a local neighborhood and determines the strength of *viscous forces* in the fluid. The relationship between strain rate and stress tensor is given by a material law that depends on the type of fluid that is modeled. In this thesis, *Newtonian*

fluids are used, which have a linear relationship between stress and strain rate similar to Hooke's law in elasticity. If the fluid is isotropic, the corresponding material law is parametrized by two independent variables μ and ζ the *shear viscosity* and the *second coefficient of viscosity*, respectively. In this case, the viscous stress tensor can be computed as

$$\tau = 2\mu\dot{\varepsilon} + \zeta \operatorname{tr}(\dot{\varepsilon})\mathbb{I}, \tag{3.9}$$

where $\mathbb{I}$ is the identity matrix. The divergence of the viscous stress tensor is

$$\nabla \cdot \tau = \nabla \cdot [2\mu\dot{\varepsilon} + \zeta \operatorname{tr}(\dot{\varepsilon})\mathbb{I}] = \nabla \cdot \left[\mu\left(\nabla \mathbf{v}^T + \nabla \mathbf{v}\right) + \zeta \operatorname{tr}(\dot{\varepsilon})\mathbb{I}\right]. \tag{3.10}$$

With the identities $\operatorname{tr}(\dot{\varepsilon}) = \operatorname{tr}(\nabla \mathbf{v}) = \nabla \cdot \mathbf{v}$ and $\nabla \cdot \left(\nabla \mathbf{v}^T + \nabla \mathbf{v}\right) = \Delta \mathbf{v} + \nabla\nabla \cdot \mathbf{u}$ that can be derived with vector calculus, Eq. (3.10) further simplifies to

$$\nabla \cdot \tau = \mu\Delta \mathbf{v} + (\mu + \zeta)\nabla\nabla \cdot \mathbf{v}. \tag{3.11}$$

Inserting this into Eq. (3.7) results in

$$\rho\left(\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v}\right) = -\nabla p + \mu\Delta \mathbf{v} + (\zeta + \mu)\nabla(\nabla \cdot \mathbf{v}) + \rho \mathbf{b}. \tag{3.12}$$

When simulating air or water, the fluid can be assumed to be *incompressible*. In this case, the property $\nabla \cdot \mathbf{v} = 0$ holds (see Eq. (3.15), which is derived later). Then the equation further simplifies to

$$\rho\left(\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v}\right) = -\nabla p + \mu\Delta \mathbf{v} + \rho \mathbf{b} \tag{3.13}$$

and finally results in the Navier-Stokes momentum equation for incompressible Newtonian fluids.

In order to realistically simulate fluids and capture the details of its motion, it is important to ensure mass conservation in addition to momentum. The conservation of mass is given by fulfilling the equation

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{v}) = 0. \tag{3.14}$$

Here, the local change of density is in balance with the density that is transported by the velocity field. The density is constant for incompressible fluids and this equation further simplifies to

$$\nabla \cdot \mathbf{v} = 0. \tag{3.15}$$

The divergence of a vector field indicates the presence of sources or sinks. Therefore, a velocity field with zero divergence cannot add or remove mass from the fluid. Furthermore, the conservation of energy is automatically satisfied if incompressibility and constant temperatures are assumed.

The complete set of PDEs describing the fluid motion is then

$$\begin{aligned} \rho\left(\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v}\right) &= -\nabla p + \mu\Delta \mathbf{v} + \mathbf{f} \\ \nabla \cdot \mathbf{v} &= 0. \end{aligned} \tag{3.16}$$

These are the *Navier-Stokes equations* for incompressible Newtonian fluids. A simplification is to omit the viscous term $\mu\Delta \mathbf{v}$ resulting in the *Euler equations*

$$\begin{aligned} \rho\left(\frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \cdot \nabla \mathbf{v}\right) &= -\nabla p + \mathbf{f} \\ \nabla \cdot \mathbf{v} &= 0. \end{aligned} \tag{3.17}$$

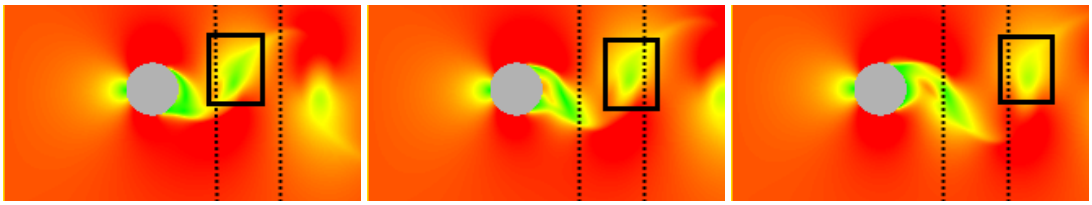


Figure 3.3.: The time series shows the evolution of the Kármán vortex street, where a circular obstacle is exposed to a flow emerging from the left hand side. The colors in the pictures represent the magnitude of velocity from low in green to high in red. Vortices are generated at the obstacle that are transported by the flow. The vertically dashed lines are placed at constant positions in the flow channel to better identify the motion of the flow and the vortices. One of the vortex cores is marked with a black box highlighting its movement.

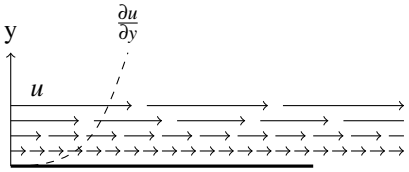


Figure 3.4.: Shear forces arise due to friction by local differences in the velocity. This example shows a close-up of a laminar channel flow and the characteristic change of velocity near the wall. The velocity gradient $\frac{\partial u}{\partial y}$ represents the only non-zero component in the strain rate tensor and is proportional to the shear forces that locally decelerate the flow.

Both types of fluid modeling, i.e., the Navier-Stokes or Euler equations, represent a coupled set of equations that must be solved simultaneously. This is reflected by the fact that the velocity appears in both equations. Furthermore, in sum there are four scalar equations and four unknown functions that need to be solved, namely three components of velocity and one pressure component.

These equations describe the motion for both types of fluid, i.e., gases and liquids. However, in the case of liquid simulations, a *two-phase modeling* has to be considered. In such a case, the flow of the liquid and the air phase is computed, and additionally the interface in between. In computer graphics applications, the simulation of the air phase is usually omitted, as the density of the air is comparatively low and only marginally affects the liquid phase. In comparison to a plain gas simulation, a *free-surface simulation* additionally requires tracking of the interface between liquid and air. This surface bounds the simulation domain Ω and usually changes during simulation.

Rearranging the momentum equation and dividing by ρ reveals which components influence the change of the velocity field

$$\frac{\partial \mathbf{v}}{\partial t} = -\mathbf{v} \cdot \nabla \mathbf{v} - \frac{1}{\rho} \nabla p + \eta \Delta \mathbf{v} + \mathbf{f}. \tag{3.18}$$

Here, $\eta = \frac{\mu}{\rho}$ is the *kinematic viscosity*. The terms on the right-hand side all contribute to the change of velocity $\frac{\partial \mathbf{v}}{\partial t}$ and correspond to different forces that act inside fluids:

a) *Advection* $\mathbf{v} \cdot \nabla \mathbf{v}$: The advection operator describes that the velocity field is transported by itself. An example for such an effect is the transport of vortices, which is depicted in Fig. 3.3.

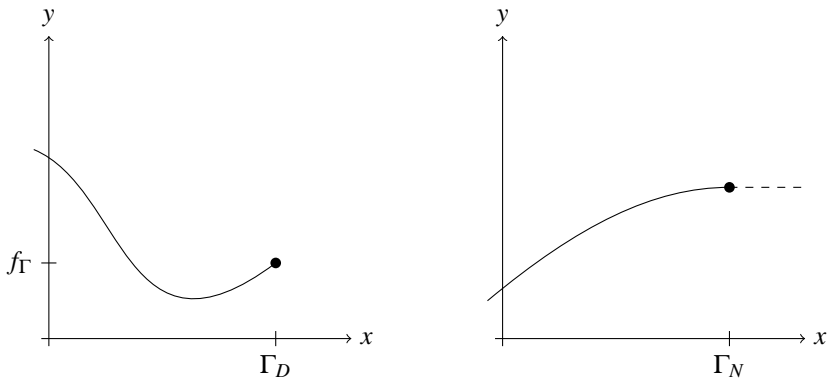


Figure 3.5.: Examples for different boundary conditions at Γ for a one-dimensional function $y = f(x)$. (a) Dirichlet boundary condition with specified value $f(\Gamma_D) = f_\Gamma$. (b) Neumann boundary condition with a specified derivative $\frac{\partial f}{\partial x}\,|_{\Gamma_N} = 0$.

b) *Pressure correction* $\frac{1}{\rho}\nabla p$: The pressure correction term implicitly accounts for mass conservation and compensates pressure differences by accelerating or decelerating velocities.

c) *Diffusion* $\eta\Delta\mathbf{v}$: The diffusion operator accounts for the friction in the fluid. Viscous forces arise when there are local differences in the velocities. A simple example showing this effect is given in Fig. 3.4.

d) *External forces* $\mathbf{f}$: External forces like gravity or any other additional force term contribute to changes in velocity.

Similarly, the rearranged Euler momentum equation is given by

$$\frac{\partial \mathbf{v}}{\partial t} = -\mathbf{v}\cdot\nabla\mathbf{v} - \frac{1}{\rho}\nabla p + \mathbf{f}. \tag{3.19}$$

Eq. (3.19) is used in the following derivations because explicit viscosity is not required as the applied numerical schemes already exhibit significant numerical viscosity.

In addition, the Navier-Stokes and the Euler equations require the specification of boundary conditions for velocity and pressure. There are two types of boundary conditions that are required for fluid simulation:

1. *Dirichlet boundary conditions* that specify the value of the function.
2. *Neumann boundary conditions* that determine the gradient of the function.

Fig. 3.5 shows one-dimensional examples for both types of boundary condition. Which kind of boundary condition is required depends on the type of scenario to be simulated. For example, in a wind tunnel scenario the inflow velocity defined at the inlet is a Dirichlet boundary condition. The normal component must be zero at obstacles or walls, as there is no flow into a solid object. If only the normal component is constrained this condition is called *slip condition* as the velocity can slip tangentially to the solid. A *no-slip boundary condition* is given if the whole velocity vector is constrained to be zero. Both, no-slip and slip conditions are Dirichlet boundary conditions that either constrain the velocity or its normal component. In addition, pressure boundary conditions are required. These need to be compatible with the velocity boundary conditions in order to ensure that the problem is well-posed. For example at the inlet or at solid boundaries the pressure gradient in normal direction needs to be zero. As the pressure gradient influences the velocity along its direction this avoids a change

in the normal component of the velocity. Hence, a Neumann boundary condition for the pressure is in this case simultaneously a Dirichlet boundary condition for the normal component of velocity. Dirichlet conditions for the pressure are required at open boundaries, i.e., for non-constrained velocities at wind tunnel outlets or at free surfaces for liquid simulation. Usually, the pressure is set to zero in these cases. If both Neumann and Dirichlet conditions are required on disjoint parts of the boundary, it is called *mixed boundary condition*.

In the following subsection, the discretization and numerical integration of the PDEs for computing the evolution of the flow are described.

3.1.2. Numerically solving the equations

Numerical techniques for the solution of the Navier-Stokes equations are also highly relevant in engineering. There, simulations are required to assess and to predict physical properties of a product before building it. The simulation methods in engineering are developed under the term *computational fluid dynamics* (CFD), where Ferziger and Peric's book [FP02] gives a good overview and introduction to standard numerical methods. In engineering applications the focus is on as accurate as possible solutions. For computer graphics, visual plausibility is more important and accuracy can be traded for computation time. However, many of the concepts are similar and various methods have been applied and exchanged between both domains. Bridson's book [Bri08] gives a good overview over fluid simulation techniques in computer graphics. The basic approach is briefly outlined in this subsection.

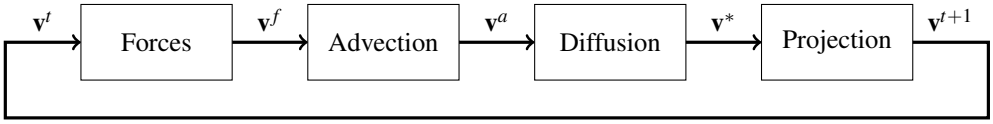


Figure 3.6.: In the fractional step method, the time integration of the velocity field $\mathbf{v}^t \rightarrow \mathbf{v}^{t+1}$ is subsequently computed by the individual contributions of the operators. To obtain the velocity $\mathbf{v}^{t+1}$ for the next time step, the external forces are applied first and then the effects of advection and diffusion are computed. Finally, $\mathbf{v}^*$ is projected in order to obtain a mass conserving velocity field $\mathbf{v}^{t+1}$.

In order to compute the motion described by the Euler equations (3.17), a numerical discretization w.r.t. space and time has to be applied. The standard approach in computer graphics is the so-called *operator splitting* or the *fractional step method*. This approach has been developed in the CFD community by Kim and Moin [KM85] and is widely adopted in computer graphics [Sta99, FSJ01, ZB05, LAF11]. There, the PDE is split and the contributions of the advection, external force and pressure correction operators are computed consecutively and in isolation as depicted in Fig. 3.6. Numerically, this corresponds to solving the following equations one after another:

$$\frac{\partial \mathbf{v}}{\partial t} = \mathbf{f}, \tag{3.20}$$

$$\frac{\partial \mathbf{v}}{\partial t} = -\mathbf{v} \cdot \nabla \mathbf{v}, \tag{3.21}$$

$$\frac{\partial \mathbf{v}}{\partial t} = -\frac{1}{\rho} \nabla p \quad \text{with} \quad \nabla \cdot \mathbf{v} = 0. \tag{3.22}$$

Finally, the velocity field in the next time step is obtained. For each of the equations, an appropriate integration scheme has to be applied, as derived in the following.

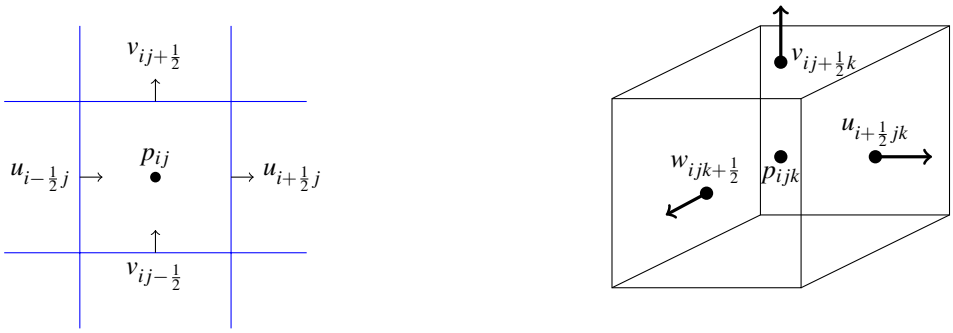


Figure 3.7.: The sketches show one cell of a staggered grid discretization in 2D and 3D (similar to [Bri08]). Pressure samples are stored at cell centers, whereas the normal components of velocity are stored at faces.

3.1.2.1. Spatial discretization

In order to solve the Eqns. (3.20), (3.21) and (3.22), a spatial discretization of the simulation domain is required. In computer graphics it is common practice to use a regular grid with equidistant spacing as depicted in Fig. 3.7. This is due to the fact that the discretization of fluid-solid or fluid-air interfaces is simplified and adaptivity is usually not required in contrast to the engineering domain. Often computer animation scenarios require handling moving geometry, which can be delicate when remeshing is required in every time step.

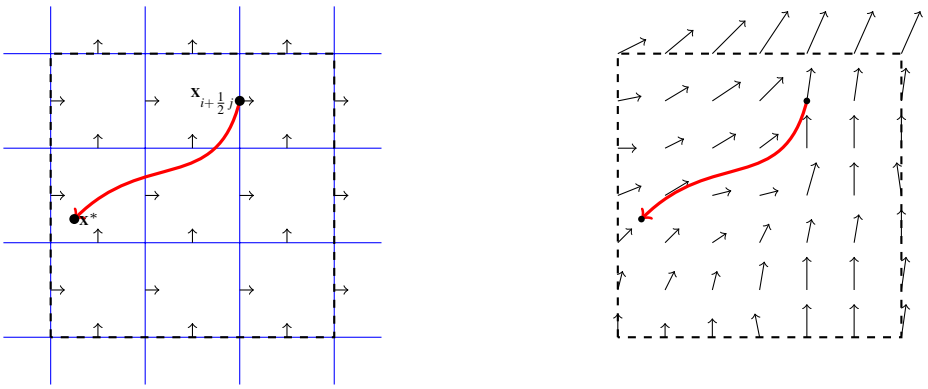
There are various options where to place the degrees of freedom for representing velocity $\mathbf{v}$ and pressure p. However, a widely adopted concept for regular grids is to apply the *staggered grid discretization* introduced by Harlow and Welch [HW65], which turns out to be numerically advantageous for the pressure correction. In this discretization, samples of the scalar field of pressure p are stored at cell centers and the normal components of the vector-valued velocity field $\mathbf{v}$ are stored at the interfaces between the cells as shown in Fig. 3.7. For a regular grid with $n_x \times n_y \times n_z$ cells, the pressure samples p_{ijk} are indexed by $i = 0,\ldots,n_x-1$, $j = 0,\ldots,n_y-1$ and $k = 0,\ldots,n_z-1$ and are located in the center of cell C_{ijk}. Furthermore, the components of the velocity vectors are given by

$$\mathbf{v} = \begin{pmatrix} u \\ v \\ w \end{pmatrix} \tag{3.23}$$

and are split for storage. In a staggered grid discretization the components of the velocity are shifted and stored at the faces of the cell, which is indicated by fractional indices, e.g., $u_{i+\frac{1}{2}jk}$, $v_{ij+\frac{1}{2}k}$ and $w_{ijk+\frac{1}{2}}$. Here, the fraction $\pm\frac{1}{2}$ in the index indicates the shift w.r.t. the corresponding coordinate axis as shown in Fig. 3.7. The state of the system is then represented by $n_x \times n_y \times n_z$ pressure samples and $(n_x+1) \times n_y \times n_z$, $n_x \times (n_y+1) \times n_z$ and $n_x \times n_y \times (n_z+1)$ samples for the components u, v and w, respectively.

3.1.2.2. Semi-Lagrangian advection

In order to solve the advection step given in Eq. (3.21), a common practice in CFD is to directly employ an FD discretization for the advection equation. However, when using such a discretization the time step needs to be limited to obtain a numerically stable solution. A standard approach in computer graphics is to use a *semi-Lagrangian advection* scheme to compute the effect of advection resulting in $\mathbf{v}_a$. There, virtual and massless



(a) (b)

Figure 3.8.: (a) Computing the update for one velocity sample defined on the grid at $\mathbf{X}_{i+\frac{1}{2}j}$ by using semi-Lagrangian advection. (b) The virtual particle is traced backwards in time through the flow of a velocity.

particles are placed at the locations of the velocity samples $\mathbf{X}$. They are traced by following the flow of the velocity field to the endpoint $\mathbf{X}^*$ backwards in time within a time step Δt. The velocity $\mathbf{v}^t(\mathbf{X}^*)$ at the endpoint of the particle trajectory is then used to update the velocity sample at the initial position $\mathbf{X}$.

Backward tracing, i.e., the application of a negative time step, results in a numerical stable time integration of the advection step. This scheme was introduced by Stam [Sta99], is adopted widely in computer animation and is equivalent to implicit time integration as velocity samples at the end of the time step are incorporated. Therefore, the position where the virtual particle originated is determined by performing particle tracing backwards in time as depicted in Fig. 3.8. In this example a particle at the location of a u-component sample $\mathbf{X}_{i+\frac{1}{2}jk}$ is traced to obtain its origin.

A particular simple integration scheme for tracing the particle is to apply an Euler step

$$\mathbf{X}^* = \mathbf{X}_{i+\frac{1}{2}jk} - \Delta t \mathbf{v}(\mathbf{X}_{i+\frac{1}{2}jk}), \tag{3.24}$$

which is backwards in time because of the negative time step. The update for the velocity at $\mathbf{X}_{i+\frac{1}{2}jk}$ is then obtained by evaluating the velocity field at $\mathbf{X}^*$

$$u_{i+\frac{1}{2}jk} = u(\mathbf{X}^*). \tag{3.25}$$

If the particle's end point is outside of the simulation domain, the boundary condition must be chosen. In both equations the current velocity $\mathbf{v}^t$ or one of its individual components needs to be determined at arbitrary positions. A standard approach is to use trilinear interpolation where nearby velocity samples and corresponding weights are determined to compute the interpolated velocity. However, employing a simple Euler step and using trilinear interpolation introduces numerical viscosity in the simulation, as sharp features are smoothed.

In order update the entire velocity field, this procedure must be repeated for all u samples in the grid and likewise for the v and w components. The accuracy of the particle tracing step in Eq. (3.24) can be increased by employing higher-order integration schemes such as *Runge-Kutta methods*. The particle tracing sketched in Fig. 3.8 is also obtained by such a higher-order scheme as the trajectory is curved and not linear in contrast to a single Euler step.

An alternative approach to increase the accuracy is to apply several advection steps to first estimate and then correct the error that would have been induced with a one-step approach. The *back and forth error compensation and correction method* (BFECC) introduced by Kim et al. [KLLR05] is such an approach to reduce the numerical viscosity in the advection step. There, the difference between a forward and a backward step is evaluated to estimate the error. In a final advection step, the computed error is considered in order to achieve second order accuracy. This concept of combining forward and backward steps was extended by Selle et al. [SFK*08]. In their approach the *unconditionally stable MacCormack method* was developed, which is conceptually similar to the BFECC approach. However, it only requires two advection steps instead of three and is second order accurate in space and time as well.

3.1.2.3. Pressure correction

Pressure correction or *pressure projection* refers to the application of the pressure gradient in Eq. (3.22) to update the velocity. However, neither the momentum nor the mass conservation equations state how to obtain the pressure p. Both equations in combination represent four scalar equations and therefore p is given implicitly. The concept of decoupling the computation for velocity and pressure has been introduced to the CFD community by Chorin [Cho67] and has been widely adopted for grid-based fluid simulation in computer animation. There, the momentum equation is integrated w.r.t. time by the fractional step method without the pressure term. The resulting velocity fields may not be mass conserving, i.e., the intermediate velocity field $\mathbf{v}^*$ is not divergence-free $\nabla \cdot \mathbf{v}^* \neq 0$. This property is recovered by applying pressure projection, which is based on the *Helmholtz decomposition* of vector fields. It states that a vector field can be decomposed into a sum of the gradient of a scalar function and the rotation of a vector field

$$\mathbf{v}^* = \nabla\Phi + \nabla \times \mathbf{a}. \tag{3.26}$$

The latter term corresponds to the application of the differential Nabla operator with the cross product and is mass conserving by definition. This is due to the fact that the divergence of the rotation of a vector field is always zero

$$\nabla \cdot (\nabla \times \mathbf{a}) = 0, \tag{3.27}$$

which is a central identity in vector calculus. Thus, the term $\nabla\Phi$ in Eq. (3.26) must be subtracted from $\mathbf{v}^*$

$$\mathbf{v} = \mathbf{v}^* - \nabla\Phi, \tag{3.28}$$

in order to *project* it onto the mass conserving vector fields. To obtain Φ, the divergence operator is applied on both sides of the Helmholtz decomposition of Eq. (3.26)

$$\nabla \cdot \mathbf{v} = \nabla \cdot \nabla\Phi + \nabla \cdot \nabla \times \mathbf{a} = \nabla \cdot \nabla\Phi. \tag{3.29}$$

This yields $\Delta\Phi = \nabla \cdot \mathbf{v}$, a *Poisson equation*, which can be solved for Φ. Here, Δ is the *Laplace operator* or the *Laplacian*, which is defined by the divergence of the gradient for scalar functions. The time discretization of the continuous pressure correction given in Eq. (3.22) is

$$\frac{\mathbf{v}^{t+1} - \mathbf{v}^t}{\Delta t} = -\frac{1}{\rho}\nabla p, \tag{3.30}$$

with velocity $\mathbf{v}^{t+1}$ at the end and $\mathbf{v}^t$ at the beginning of a time step. Rearranging this equation to

$$\mathbf{v}^{t+1} = \mathbf{v}^t - \frac{\Delta t}{\rho}\nabla p, \tag{3.31}$$

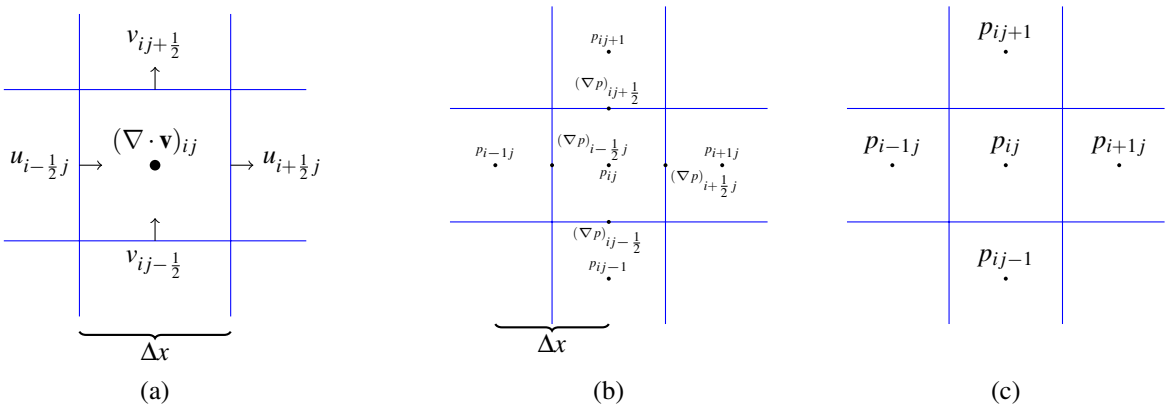


Figure 3.9.: (a) The discrete divergence located at cell centers is computed by the normal components of the velocity, which are stored at cell faces. (b) The pressure gradients are defined at cell interfaces and are evaluated by computing the central difference between adjacent pressure samples. (c) Pressure variables from adjacent cells are involved in the discretization of the Laplacian.

the similarity to Eq. (3.28) is obvious, where Φ corresponds to $\frac{\Delta t}{\rho} p$. In summary, the pressure for Eq. (3.22) can be computed by solving the Poisson equation

$$\nabla \cdot \nabla p = \frac{\rho}{\Delta t} \nabla \cdot \mathbf{v}^*, \tag{3.32}$$

on the domain Ω with intermediate velocity $\mathbf{v}^*$. The projected velocity can then be obtained by subtracting the pressure gradient from $\mathbf{v}^*$

$$\mathbf{v} = \mathbf{v}^* - \frac{\Delta t}{\rho} \nabla p. \tag{3.33}$$

The computation of pressure by solving Eq. (3.32) requires the specification of appropriate boundary conditions at the simulation domain border Γ. If the fluid is adjacent to a non-moving, solid boundary with normal vector $\mathbf{n}$, the derivative of the pressure, i.e., a homogeneous Neumann boundary condition, i.e.,

$$\frac{\partial p}{\partial \mathbf{n}} |_{\Gamma_N} = 0 \tag{3.34}$$

has to be specified, where Γ_N is the Neumann boundary. This ensures that the normal component of the velocity does not change when subtracting the pressure gradient. If the fluid boundary is a free surface, a Dirichlet condition $p |_{\Gamma_D} = 0$ at the Dirichlet boundary Γ_D is specified.

In order to numerically compute p and the projected velocity $\mathbf{v}$, the Poisson equation (3.32) and $\nabla \cdot \mathbf{v}^*$ are discretized by applying an FD scheme. The staggered grid depicted in Fig. 3.7 shows the sample locations of p and $\mathbf{v}$ with the components u, v and w. The divergence of the intermediate velocity field is then determined by computing a central difference

$$(\nabla \cdot \mathbf{v})_{ijk} = \frac{u_{i+\frac{1}{2}jk} - u_{i-\frac{1}{2}jk} + v_{ij+\frac{1}{2}k} - v_{ij-\frac{1}{2}k} + w_{ijk+\frac{1}{2}} - w_{ijk-\frac{1}{2}}}{\Delta x}, \tag{3.35}$$

where Δx is the grid spacing and the divergence is stored at cell centers as shown in Fig. 3.9 (a). The evaluation of the pressure gradient with central differences is given by

$$(\nabla p)_{i+\frac{1}{2}jk} = \frac{p_{i+1jk} - p_{ijk}}{\Delta x}, \tag{3.36}$$

where the location of the pressure gradient is at the cell interfaces. This equation represents the evaluation of the pressure gradient in x-direction at the velocity sample $u_{i+\frac{1}{2}jk}$. The other components can be computed similarly, which is depicted in Fig. 3.9 (b). The discretization of the Laplacian of p is obtained by applying the divergence operator to Eq. (3.36)

$$(\Delta p)_{ijk} = (\nabla \cdot \nabla p)_{ijk} = \frac{p_{i+1jk} + p_{ij+1k} + p_{ijk+1} - 6p_{ijk} + p_{i-1jk} + p_{ij-1k} + p_{ijk-1}}{\Delta x^2}, \tag{3.37}$$

Combining Eq. (3.35) and Eq. (3.37) yields the discretization for one cell C_{ijk} of the Poisson equation (3.32) on regular grids

$$\begin{aligned} &\frac{-p_{i+1jk} - p_{ij+1k} - p_{ijk+1} + 6p_{ijk} - p_{i-1jk} - p_{ij-1k} - p_{ijk-1}}{\Delta x^2} \\ &= -\frac{\rho}{\Delta t} \frac{u_{i+\frac{1}{2}jk} - u_{i-\frac{1}{2}jk} + v_{ij+\frac{1}{2}k} - v_{ij-\frac{1}{2}k} + w_{ijk+\frac{1}{2}} - w_{ijk-\frac{1}{2}}}{\Delta x}, \end{aligned} \tag{3.38}$$

where the equation has been multiplied by -1 to obtain a positive semi-definite system. Combining all cells results in a large sparse linear system that can be solved by, e.g., a CG solver to compute the pressure p. For cells at the boundary of the simulation domain the equations must be modified to account for the specified boundary conditions. To finally obtain the projected velocity field, the pressure gradient in Eq. (3.36) has to be evaluated and subtracted from initial velocity field.

3.1.2.4. Time integration

The computations required to simulate the motion of a fluid will be summarized here. The subsequent states of the velocity are obtained by applying a time integration scheme to Eqns. (3.20), (3.21) and (3.22). The sequence for computing the update of velocity from time t to $t+1$ by employing the fractional step is (also shown in Fig. 3.6):

1. The change of the current velocity $\mathbf{v}^t$ due to external forces can be determined by a simple difference equation
$$\mathbf{v}^f = \mathbf{v}^t + \Delta t \mathbf{f}^{\text{ext}}. \tag{3.39}$$
2. The effect of advection is computed by applying the algorithm described in Subsection 3.1.2.2 to obtain the advected velocity $\mathbf{v}^f \rightarrow \mathbf{v}^*$.
3. Finally, the intermediate velocity $\mathbf{v}^*$ is projected as described in Subsection 3.1.2.3 to account for the pressure gradient and to conserve mass.

For fluids with low viscosity like water or air, it is possible to omit the computation of diffusive effects. This can be done as the advection scheme already induces numerical viscosity. In this case, the velocity resulting from advection is directly $\mathbf{v}^*$ instead of $\mathbf{v}^a$. If simulation of viscous effects for highly viscous materials is required, an additional linear system must be solved to compute $\mathbf{v}^a \rightarrow \mathbf{v}^*$ for unconditional stability. The details of how to compute this effect are described in Bridson's book [Bri08].

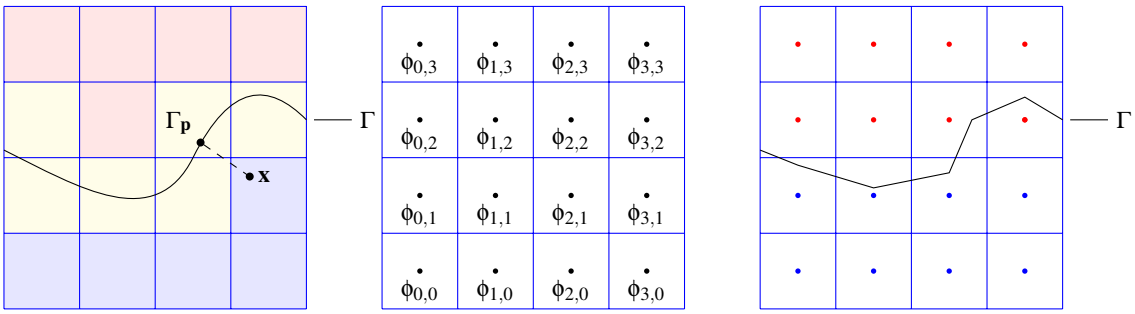


Figure 3.10.: (a) The interface Γ decomposes the simulation domain into active (blue and yellow) and inactive cells (red). The free surface passes through the yellow cells that require special handling as they are on the boundary. The signed distance function associated with Γ encodes the distance of every point $\mathbf{x}$ to the closest point Γ_p on the interface. (b) Discretization of the signed distance function ϕ as cell-centered samples ϕ_{ij}. (c) Example of a discretized signed distance function and its discrete level set Γ. Values $\phi_{ij} > 0$ are colored in red whereas $\phi_{ij} < 0$ are colored on blue. The marching squares algorithm, the two-dimensional equivalent to marching cubes, generates the discretized interface, i.e., the line segments.

3.1.2.5. Free-surface simulation

In order to simulate liquids with a moving *free surface*, the algorithm described in the previous subsection must be adapted. An initially specified interface between liquid and air has to be tracked over time by determining how it deforms under the influence of the velocity field. In addition, the interface or free surface Γ represents the boundary of the simulation domain Ω. Therefore, the continuous change of the simulation domain with corresponding change of boundary conditions has to be taken into account.

A standard approach in computer graphics proposed by Osher and Fedkiw [OF02] is to use a *level set* to represent the interface between liquid and air. A level set is an iso-contour of a scalar function defined on the domain Ω and implicitly defines the interface. In the following, a *signed distance function* is used as the scalar function, as proposed by Bridson [BMF07]. The signed distance function is given by

$$\phi(\mathbf{x}) = \begin{cases} -\|\Gamma_{\mathbf{p}} - \mathbf{x}\| & \text{if } \mathbf{x} \in \Omega \\ \|\Gamma_{\mathbf{p}} - \mathbf{x}\| & \text{if } \mathbf{x} \notin \Omega, \end{cases} \tag{3.40}$$

where $\Gamma_{\mathbf{p}}$ is the closest point to $\mathbf{x}$ on the interface Γ as shown in Fig. 3.10 (a). Eq. (3.40) defines negative distances for points located inside the simulation domain and positive distances for positions outside.

In order to numerically compute the evolution of the interface, a discretization of the signed distance function is required. One option is to utilize cell-centered samples ϕ_{ijk} as shown in Fig. 3.10 (b) for a two-dimensional example. This implicit representation must be transformed into an explicit representation to determine the boundary conditions and for rendering purposes. This can be achieved by using the marching cubes approach [LC87], which extracts iso-surfaces of implicit functions on regular grids. Fig. 3.10 (c) shows an example of a discretized signed distance function and its explicit representation. There, positive values of ϕ_{ij} are shown as red dots and negative as blue dots resulting in a set of line segments for Γ in this two-dimensional case.

The interface is advected by the velocity field. This change can be computed by updating the discrete samples ϕ_{ijk} with a semi-Lagrangian approach as described in Section 3.1.2.2. There, massless particles at cell centers C_{ijk} are traced backwards in time. At the end point the new value for ϕ_{ijk} is determined by interpolation. This is similar to the approach for velocity advection described in Section 3.1.2.2. Here, cell centers are the start positions for particles and the discretized signed distance field is used for interpolation. Due to numerical drift, this advection scheme does not maintain the property of being the signed distance function ϕ. Therefore, the signed distance field with all ϕ_{ijk} must be recomputed regularly as described by Bridson [BMF07].

Furthermore, the advection of the velocity field might access velocity samples that are outside the simulation domain. Evaluating the velocity close to the interface can give a distorted result, as velocity samples of inactive cells might not be valid. Therefore, velocity samples for inactive cells in the proximity of the interface are determined by constant extrapolation from valid cells.

The pressure correction described in Subsection 3.1.2.3 must be adapted to account for the interface Γ. More specifically, the interface represents a Dirichlet boundary condition Γ_D, where the pressure $p = 0$ is specified. For inactive grid cells the corresponding row in the linear system is set to zero except the diagonal value which is set to one. Furthermore, if an active cells is adjacent to a inactive cell, e.g., C_{i-1jk}, the corresponding condition $p_{i-1jk} = 0$ is inserted into Eq. (3.38). In order to achieve higher accuracy, the exact interface position $\mathbf{x}_I$ can be considered when setting up the linear system. The *ghost fluid method* (GFM) introduced by Gibou et al. [GFCK02] is such an approach, where a ghost value

$$p^G_{i+1jk} = \frac{\mathbf{x}_I - (\theta - 1)\mathbf{x}_{ijk}}{\theta} \tag{3.41}$$

outside the simulation domain is determined by extrapolation and inserted into Eq. (3.38). In this example, cell C_{i+1jk} does not belong to the simulation domain, cell C_{ijk} is at the boundary and the fraction

$$\theta = \frac{\mathbf{x}_I - \mathbf{x}_{ijk}}{\Delta x}, \tag{3.42}$$

is determined by the interface position $\mathbf{x}_I$ and the cell center $\mathbf{x}_{ijk}$. With this formulation, the correct interface position is taken into account.

In summary, the algorithm for simulating liquid is as follows:

1. Compute velocity values for inactive cells near the interface by constant extrapolation
2. Compute velocity update due to external forces
3. Advect velocity field $\mathbf{v}$
4. Advect the signed distance function ϕ
5. Generate a surface representation with the marching cubes algorithm
6. Update the linear system for the pressure correction according to changes of the simulation domain
7. Apply pressure correction

In comparison to the simulation of gas described in the previous section, additional steps have to be performed. The extrapolation step is required to not distort the advection near the interface due to numerical drift and time step resolution. In order to track the moving free surface, the signed distance function must be advected and an explicit representation has to be generated. Furthermore, the linear system must be updated as the simulation domain and the boundary change due to the evolution of the free surface.

3.2. Related work

One goal of this thesis is to improve the quality and the speed of fluid simulations. The contributions to achieve this goal are classified into three pillars

- *more accurate discrete representations*
- *efficient methods for linear systems*
- *data structures and methods for massively parallel computing*

This classification is reflected in this section when reviewing the related work in the field of computer graphics. At first, the related work covering visual plausible fluid simulation is summarized. Section 3.2.1 recapitulates early work and important methods that aim at realistically reproducing the motion of fluids for computer graphics applications. In Subsection 3.2.1.1, approaches are described that solve the Navier-Stokes equations on Eulerian regular grids. These methods constitute the basis of the novel approaches developed in this thesis. Alternative methods for realistically computing the motion of fluids are briefly discussed in Subsection 3.2.1.2. In Section 3.2.2, related work is summarized that focuses on regular grid simulations with *embedded geometry*. Embedded geometry methods resolve solids or free surfaces that are not aligned with grid. These sub-grid accurate methods essentially differ in the definition of boundary conditions and are related to the first pillar. Section 3.2.3 discusses multigrid approaches for efficiently solving linear systems arising from pressure correction. This highly efficient class of algorithms is usually the best choice for high resolution simulation because of their linear time complexity and is associated with the second pillar. In Section 3.2.4, the focus is on fast methods for fluid simulation. These type of approaches focus on interactivity and aim to compute simulation steps while the result is visualized. The methods are mainly massively parallel approaches that are executed on GPU hardware and are related to the third pillar.

3.2.1. Fluid simulation for computer graphics

This section reviews the major developments in physically based fluid simulation in computer graphics. The presented methods aim for visually plausible simulations. In the following subsection, methods are summarized that solve the Navier-Stokes or Euler equations on regular grids. These constitute the basis for the methods developed in this thesis. Subsection 3.2.1.2 addresses alternative approaches for generating visual plausible animation.

3.2.1.1. Fluid simulation on regular grids

Although some early works in computer graphics mimic the motion of fluids by using geometric or procedural approaches, relying on a physical formulation generally results in higher realism. Usually, the *incompressible Navier-Stokes equations* or the *incompressible Euler equations* are employed that have been derived and described in Section 3.1.1. Solving these equations accurately and efficiently is a central topic in the CFD community. Ferziger and Peric's [FP02] text book summarizes basic methods in that area. In CFD, many of methods have been developed and some of them significantly influenced computer graphics. The most relevant CFD works to this thesis are discussed in the following but a large amount of the specialized literature related to engineering is not reviewed here. In computer graphics, there is also a vast amount of literature. The state of the art report by Nealen et al. [NMK*06] dedicated a chapter to Eulerian fluid simulation and provides a summary of developments. Additionally, Bridson's book [Bri08] provides a good overview of numerical techniques for fluid simulation in computer graphics.

One of the starting points for computer graphics that was developed in the CFD community is the work by Harlow and Welch [HW65]. They utilized a Eulerian regular staggered grid discretization to solve the Navier-Stokes equations. In this type of discretization the storage of pressure samples is at cell centers and normal components of the velocity on cell faces resulting in numerically stable discretizations. Furthermore, they introduced the so-called *particle in cell* (PIC) method that uses *marker particles* to track the free surface of a liquid. Another standard approach that has been developed in CFD is the *fractional step method*. There, the individual operators of the PDE are computed one after another and in isolation. This method has been introduced by Kim and Moin [KM85] and is widely adopted in computer graphics.

The first physically based approach in computer graphics for fluid simulation was presented by Kass and Miller [KM90]. In their method, they simulate the dynamics and propagation of waves on water surfaces. They use the *shallow water equations*, which are a two-dimensional approximation of the Navier-Stokes equations with the height of the water surface as primary unknown function. For moderately sized discretizations they achieve interactive rates. Foster and Metaxas [FM96] extended this approach to a full three-dimensional simulation based on the Navier-Stokes equations. They apply *explicit time stepping*, which is restricted to comparably small time steps. The equations are discretized with an FD scheme on a staggered grid. To compute the pressure correction, a Jacobi-like solver with only a few iterations is utilized. This generates visual plausible results but does not solve the linear system accurately. Furthermore, they track the water surface by using marker particles. In a follow-up work, Foster and Metaxas [FM97] adapted this approach to simulate the motion of gas. Additionally, the temperature of the gas is simulated and thermal buoyancy induced by temperature differences generates additional forces, which result in realistic swirly motion. Again, the pressure solution is computed with a Jacobi-like relaxation scheme using only a few iterations.

All of these methods are limited by a maximum admissible time-step size to achieve numerical stable simulations, as explicit time stepping is employed. This restriction was overcome by the so-called *semi-Lagrangian advection* introduced by Stam [Sta99]. Instead of using only an Eulerian representation, where the values are stored on the grid, Lagrangian particles are used to compute the effect of advection. In order to achieve unconditional stability, particles are traced backwards in time, which is equivalent to an implicit time integration. At the particle's end position the velocity is determined by trilinear interpolation, which is similar to the *method of characteristics* earlier developed in CFD by Courant et al. [CIR52]. Although, the proposed method is unconditionally stable, it tends to generate additional numerical viscous effects. This is due to interpolation, which smooths out sharp features. However, these features account for the realism of the fluid motion, as, e.g., small vortices are likely to vanish. This is especially problematic when simulating low viscosity fluids like water or air. Therefore, the numerical viscosity of the advection step is one of the major weaknesses of Stam's method. To counteract this effect, many works have been presented addressing and mitigating the numerical viscosity in the semi-Lagrangian advection step. Fedkiw et al. [FSJ01] proposed a *vorticity confinement* scheme to introduce rotational forces to recreate rotational motion. They solve the Euler equations directly without the viscous term, as semi-Lagrangian advection already introduces a fair amount of viscous effects. As a further extension, a cubic interpolation scheme is proposed for local evaluation of velocity. Zhu and Bridson [ZB05] introduced the *Fluid-Implicit-Particle (FLIP) approach* into the computer graphics community, which extends the PIC method proposed by Harlow and Welch [HW65]. The idea of FLIP is to transfer differences instead of actual values as used in the PIC approach. This completely eliminates numerical viscosity, but may introduce other artifacts such as noise due to artificial oscillation. Zhu and Bridson use this approach to simulate the fluid-like behavior of sand by augmenting a standard fluid solver with frictional effects within the material and on the boundary to simulate granular material.

Besides the simulation of gas, simulations using the fractional step method with semi-Lagrangian advection and pressure correction are also used for the animation of liquids. These simulation scenarios are called *two-phase flows*, because different phases, i.e., liquid and gas are modeled. Usually, the simulation of air is not carried

out. A major difference to gas simulations is that the border between liquid and air, the *free-surface* has to be tracked over time. The methods summarized in the following mainly differ in how the free surface is modeled and tracked. Early work like the methods presented by Harlow and Welch [HW65] or Foster and Metaxas [FM96] use marker particles that are incorporated in the semi-Lagrangian advection to track the free surface. Particles are spawned within the liquid phase and used to follow its evolution. After advection this information is used to determine if a cell contains fluid or belongs to the air phase. Drawbacks of these methods are their inability to conserve volume and the difficulty to reconstruct a smooth surface for visualization. As an alternative from CFD, Osher and Sethian [OS88] proposed to use a *level set function* for tracking. A level set is the iso-contour of a volumetric scalar function which is advected by the fluid flow. This function is discretized on a grid and updated with semi-Lagrangian advection. One of the major advantages is that the resulting surface can be generated without artifacts by utilizing a marching cubes algorithm. In a series of papers, Enright et al. [EMF02, EFFM02, EF02] improved the tracking of the surface by proposing a hybrid method which combines marker particles and level sets to capture the interface between fluid and air. They use higher-order advection schemes to track both types of representations. The so-called *particle level set method* effectively tracks the surface in contrast to the volume as in particle-based approaches. The particles are put on each side of the level set and are also advected to effectively enhance the resolution of the level set. Furthermore, they propose to extrapolate the velocity field from the fluid to the air in order to provide a smoother transition across the interface. This reduces artifacts when applying semi-Lagrangian advection. A similar approach has been presented by Bargteil et al. [BGOS06]. They use a polygonal surface tracking method called *semi-Lagrangian contouring*. They use a signed distance function and advect it backwards in time as this representation avoids self-intersections by construction. In every time step a new polygonal surface is then generated by contouring the zero set of the signed distance field using a marching cubes approach. However, a drawback of this method is that small, sub-grid details are lost.

An alternative to particles and level sets is to explicitly track the border between liquid and air by a polygonal surface representation. In this context, a naive approach of directly advecting the vertices of the polygonal mesh leads to tangling and self-intersections and therefore to non-correctable artifacts. In the work of Müller [Mül09], a simplistic surface tracking approach is presented, where the vertices of the triangle mesh are advected directly. In order to maintain a valid discretization, self-intersections are detected and fixed by locally applying the marching cubes algorithm. Wojtan et al. [WTGT09] introduced a method that uses triangular meshes instead of an implicit function as primary representation of the liquid interface. The problems with tangling and self-intersection are solved by computing an intermediate signed distance field to detect the need for topological changes of the mesh. This information is then used to perform merge and split operations on the triangle mesh data structure. Brochu and Bridson [BB09b] followed a similar approach to avoid the self-intersecting artifacts. However, they directly perform collision detection on the moving mesh to identify the need for merge or split operations. Later, Brochu et al. [BBB10] presented a liquid simulation on a *Voronoi cell discretization* of pressure samples. They use a polygonal mesh to track the free surface and capture small scale details accurately by placing the pressure samples near the liquid surface. Furthermore, they employ a finite volume (FV) discretization and derive special boundary conditions for the Voronoi cells to capture the precise position of the surface in the pressure correction step. A survey of the developments regarding explicit mesh-based surface tracking techniques is given in the SIGGRAPH course by Wojtan et al. [WMFB11].

Boyd and Bridson [BB12] computed a real two-phase simulation by additionally considering the velocity field of the air phase. Two separate velocity fields are then simulated and combined in the pressure correction step to obtain global mass conservation. They introduce the *MultiFLIP method* that associates particles either to the air or the fluid phase to accurately reconstruct the interface with low numerical viscosity. This results in a more realistic motion that is able to reproduce detailed two-phase flow phenomena like bubbles. Bojsen-Hansen and Wojtan [BHW13] developed a method for eliminating artifacts that arise when combining a high-resolution surface mesh with a simulation on a low-resolution grid. To this end, an error function is designed that penalizes

non-physical or non-compatible states of the mesh w.r.t. to the coarse discretization. They reduce the error with a gradient descent technique and remove high-frequency surface noise that typically arises when the grid resolution is too low.

Da et al. [DBG14] extended the approach of mesh-based surface tracking to multimaterial interfaces. With this technique they are able to capture complex topological changes with intersection-free meshes for multiple liquids. They can simulate mixtures as well as non-mixing liquids when high surface tensions are present. Edwards and Bridson [EB14] utilized a higher-order *discontinuous Galerkin method* (DGM) for the discretization of the Poisson equation and couple it to a surface tracking approach. They use p-adaptivity with high polynomial degrees near the surface to resolve small scale details with a coarse mesh discretization. Zhu et al. [ZQC*14] presented an approach to capture droplets, filaments, thin films and volumes of liquids by using a so-called *codimensional mesh*. This mixed mesh models disconnected points, non-manifold edges and triangles to represent thin structures. This representation allows for interesting surface tension effects such as the motion of a soap bubble and even its burst into small droplets.

The methods reviewed in this subsection form the basis and serve as a reference for the approaches developed in this thesis. The previous works mostly improve the modeling or develop techniques for new effects. These are complementary to the goals of this thesis. The proposed extensions may be combined with the methods developed here. However, computational speed is not the main goal of the methods summarized in this subsection. In many cases they are employed as an offline simulation to compute the animation of fluid dynamics. Usually, time-intensive parts such as solving the linear system for the pressure correction are computed with standard algorithms, e.g., conjugate gradient methods (CG) with diagonal (PCG) or incomplete Cholesky preconditioning (ICCG), which are comparatively slow. In order to achieve interactivity with high-resolution discretizations, novel methods are required that accelerate the solution process of the linear systems. Furthermore, distributing the computations on many processors is an option to significantly increase the ratio between accuracy and computation time. Especially massively parallel approaches for fluid dynamics are necessary to enable highly detailed simulation at interactive rates.

3.2.1.2. Alternative approaches to fluid simulation

An adaptive approach using an octree data structure for fluid simulation in computer graphics was presented by Losasso et al. [LGF04]. They increase the resolution of the octree depending on the geometric error of the solid representation or the vorticity concentration for smoke simulation scenarios. In order to obtain symmetric positive definite linear systems, they modify the standard discretization scheme to account for the non-trivial neighborhoods at T-junctions in the octree representation. They perturb the position of the pressure samples away from the cell centers such that the resulting matrix is symmetric. As the octree data structure is well suited for spatial search, the semi-Lagrangian tracing of particles can be adapted without significant additional cost.

An alternative approach to better resolve geometric details is to use tetrahedral meshes that locally conform with the geometry. Feldman et al. [FOK05] based their discretization on regular grids and propose to switch to a conforming tetrahedral mesh near solid boundaries. They show how to incorporate this hybrid mesh representation with the standard semi-Lagrangian advection and pressure correction. For the numerical discretization of the equations on tetrahedra they generalize staggered grids and store velocity samples at faces and pressure samples at cell centers. Klingner et al. [KFCO06] proposed to use a purely tetrahedral mesh discretization for the simulation of fluid flow. Additionally, they consider moving obstacles by remeshing the whole simulation domain to completely conform to the boundary. Furthermore, they present a generalization of the semi-Lagrangian advection for constantly changing meshes. In the work of Chentanez et al. [CFL*07] a tetrahedral mesh based on a body centered lattice was used to discretize the free surface of a liquid simulation. Depending on the geometric error of the free surface representation the tetrahedral mesh is regenerated. Furthermore, they propose to use an

alternative discretization and store pressure samples at the elements' barycenters instead of their circumcenters. Batty et al. [BXH10] built upon this approach but use fixed tetrahedral meshes near the boundaries and propose tetrahedral cut-cells to more accurately resolve the fluid flow. Ando et al. [ATW13] employed a tetrahedral mesh discretization with spatial adaptivity for liquid simulation. Instead of using the standard FVM, they proposed an alternative approach which minimizes the kinetic energy with pressures at nodes and velocities at barycenters. Additionally, they introduced the FLIP method on tetrahedral meshes for the advection term resulting in low numerical viscosity.

Kim et al. [KTJG08] separated large and small scale detail and augment rather low resolution simulations with turbulence. A wavelet decomposition determines the location and energy characteristics of turbulent motion, which is synthesized during the simulation run to enrich detail. Pfaff et al. [PTSG09] proposed to precompute the flow of artificial turbulent boundary layers around objects and to inject these effects during the simulation in order to achieve sub-grid fluid motion. Mullen et al. [MCP*09] presented an energy preserving integration on a tetrahedral mesh discretization. They base their numerical discretization on discrete exterior calculus (DEC), which is a scheme that is able to reproduce identities from vector calculus on a discrete level. In this formulation the artificial viscosity can be controlled by construction and a system of nonlinear equations must be solved for time integration. Sin et al. [SBH09] introduced a point-based method for simulating incompressible flow. They use particles that carry velocity information and advection is performed by collecting nearby velocity samples. Pressure correction is then conducted by constructing a Voronoi diagram of the samples and the Poisson equation is solved by using an FV approach. Lentine et al. [LZF10] used a *coarse grid projection* to accelerate the pressure correction mainly for smoke simulations. While retaining a high resolution for advection, a coarse grid for the discretization of the Poisson equation is used to obtain significantly smaller linear systems. Ando et al. [ATW15] extended this approach for liquid simulations by additionally incorporating the signed distance function in the coarse grid projection in order to reduce artifacts.

Zhu et al. [ZLC*13] introduced a far-field data structure for domain extension. They propose to use a regular grid discretization with equidistant sizing near the center of the simulation domain and increase cell sizes by power of two factors in outer regions. The adaptive discretization ensures symmetric and positive definite linear systems and allows the simulation of significantly larger domains with moderate computational overhead. Furthermore, this approach can avoid artifacts such as the reflection of waves from the simulation boundary by using large domains. English et al. [EQYF13] presented a *chimera grid embedding* for large scale simulation. A set of fine regular grids is embedded into a coarse grid discretization to locally provide a higher resolution. The fine grids can be placed arbitrarily to decompose a large scene with local adaptivity. Algorithms are presented on how to couple the overlapping regions in order to generate a global large scale fluid simulation.

Besides grid-based Eulerian methods, there is a family of purely *Lagrangian* approaches that are based on particles. *Smoothed particle hydrodynamics* (SPH) is an alternative method to simulate fluids. There, the physical quantities like velocity and mass are associated with particles that are tracked by the fluid flow without the need for computational meshes. Recent SPH developments are summarized in the state of the art report of Ihmsen et al. [IOS*14]. The approach of *position-based fluids* introduced by Macklin and Müller [MM13] is a further alternative to mimic fluid motion with a non-physical, Lagrangian approach. Another family of methods for simulating fluid flow is based on the vorticity formulation of the Navier-Stokes equations. In this context, many approaches have been presented such as using *vortex particles* (Selle et al. [SRF05]), the development of vorticity conserving methods (Elcott et al. [ETK*07]) or by employing vortex filaments to represent smoke (Weißmann and Pinkall [WP10]). In this thesis however, the focus is on the physically based, original form of the Navier-Stokes equations in the Eulerian perspective.

The methods summarized in this section are further extensions and alternative formulations to simulate visually plausible fluids. They also might benefit from the fast methods developed in this thesis.

3.2.2. Sub-grid accurate methods

Employing a regular grid discretization for fluid simulation is attractive, as the algorithms are easy to implement. Furthermore, such a discretization usually results in high performance because of cache coherent memory access. In a standard regular grid implementation, the state of each cell is marked as an obstacle, fluid or air to distinguish the different media. However, this approach is limited in resolving geometric details as a voxelized representation and cannot capture geometry which is not aligned with the discretization. In order to counteract this effect, sub-grid modeling can improve the visual quality of the simulation by adapting the discretization for embedded boundaries. These types of methods can be associated with the pillar *more accurate discrete representation* of this thesis. Sub-grid modeling extends the regular grid formulation with so-called *irregular* or *immersed boundaries*. The representation of solid geometry or free surfaces that are not aligned with the grid cells is enhanced in the simulation by explicitly incorporating the geometric border in the discretization. In particular, the Poisson equation in the pressure correction requires special treatment by adapting the boundary conditions of the pressure field. The following methods can be classified according to mainly three different types of discretization: FD, variational and FV approaches.

Fedkiw et al. [FAMO99] introduced the *GFM* based on FDs for immersed boundaries in the pressure computation. They handle jump conditions and Dirichlet boundary conditions by incorporating ghost cells outside of the fluid domain. Jump boundary conditions must be specified when discontinuities of the pressure field can be expected. Two-phase simulations are an example, where these kind of discontinuities usually arise between the liquid and gas phase. They transfer the corresponding values to the cells on the other side of the interface and achieve first order accuracy w.r.t. grid spacing. Gibou et al. [GFCK02] specialize the GFM for Dirichlet conditions using FDs to obtain second order accuracy. They propose to use linear extrapolation for the ghost cells, so that the boundary conditions at interface positions are exactly met. Both discretizations result in symmetric matrices. However, the approaches are only able to handle Dirichlet conditions for embedded geometry and cannot provide sub-grid accuracy for Neumann conditions.

A variational approach for discretizing embedded geometry has been presented by Batty et al. [BBB07]. In their method, volume fractions between adjacent cells are computed to represent a fractional coverage of solid and fluid to improve the voxelized pressure computation. The resulting scheme can be interpreted as a minimization of the total kinetic energy in the system. Their method is able to handle Dirichlet boundaries by incorporating the GFM and Neumann boundary conditions due to volume fractions. However, the accuracy of the divergence-free velocity is only first order in the L_1 and not convergent in the L_∞ norm for the velocity field as analyzed by Ng et al. [NMG09].

Johansen and Colella [JC98] presented an FV discretization to solve embedded Dirichlet boundary conditions on a two-dimensional regular grid. The resulting scheme is second order, but it generates non-symmetric linear systems that are challenging to solve. This scheme has been extended to three dimensions in the work of Schwartz et al. [SBCL06]. Ng et al. [NMG09] also use an FV discretization and focus on Neumann boundary conditions. In their approach, embedded geometry is represented as a level set function. Therefore, they propose an alternative representation that uses the fractional solid and fluid coverage at interfaces between the cells to cope with irregular domains. Using face instead of volume fractions improves the accuracy of the velocity after the pressure correction.

The methods summarized here focus on solving the Poisson equation on regular grids with embedded geometry. Methods based on FDs are not able to handle Neumann boundary conditions well. The variational approach proposed by Batty et al. [BBB07] is able to combine the GFM for Dirichlet conditions and Neumann boundaries by volume fractions. However, the accuracy of the resulting velocity field is low near boundaries. Methods based on FVs can handle Neumann conditions well, but result in non-symmetric linear systems when imposing Dirichlet conditions on embedded geometry.

There is currently no method that derives high sub-grid accuracy and results in numerically advantageous symmetric linear systems for both types of boundary conditions. Especially, high accuracy at boundaries w.r.t. different resolutions is important to construct highly convergent multigrid solvers.

3.2.3. Multigrid approaches for pressure correction

The solution of the linear system for enforcing incompressibility remains the main bottleneck in fluid simulations, especially for highly detailed simulations on large grids as shown in Table 3.1 in Section 3.3. Therefore, this section reviews methods that can be associated with the pillar *efficient methods for linear systems*. Multigrid methods theoretically provide the best performance for a large number of degrees of freedom. The books of Trottenberg and Schuller [TS01] and Briggs et al. [BHM00] provide a good overview of multigrid techniques. However, the construction of highly efficient multigrid schemes is challenging and optimal convergence rates are difficult to achieve especially for embedded geometries. Various methods have been proposed in the context of CFD or applied mathematics. The works of Johansen and Colella [JC98] and Schwartz et al. [SBCL06] apply an FV discretization for embedded Dirichlet boundary conditions. They solve the corresponding non-symmetric linear systems with a multigrid approach. A similar method has been developed by Wan and Liu [WL04], who proposed a numerical multigrid scheme based on FDs and FEs to solve irregular boundary conditions represented by a level set. However, these approaches only handle Dirichlet boundary conditions.

In the context of fluid simulation for computer graphics, McAdams et al. [MST10] presented a multigrid method embedded in a CG solver for solving Poisson equations for the pressure correction. They use a single multigrid V-cycle for preconditioning, as their multigrid method does not converge on its own. Chentanez and Müller [CM11a] analyzed McAdams' approach and stated that the divergence of the solver is due to not using a solid fraction method, as it leads to incompatible discretizations at different levels of the mesh hierarchy. They extend the approach and propose a multigrid solver for collocated discretizations (i.e., pressure and velocity samples are stored at cell centers) based on the variational approach described by Batty et al. [BBB07]. In Chentanez and Müller [CM11b] they proposed a similar multigrid method but on a variational staggered grid discretization.

Ferstl et al. [FWD14] introduced a method for liquid simulation using a finite element (FE) discretization on adaptive regular grids. They use an octree to discretize the space accurately at liquid-air boundaries and provide coarse grid cells in regions which are covered entirely by fluid. To resolve the liquid-air boundaries, a specialized marching cubes algorithm is applied to create tetrahedral elements for partially filled cells. On coarser levels cells and vertices are locally duplicated for topologically separated domains to improve convergence. However, only free surfaces, i.e., Dirichlet boundaries, are considered as embedded geometry and only grid-aligned Neumann conditions can be handled. Setaluri et al. [SABS14] presented a specialized sparse grid data structure by means of pyramids of uniform grids, where cells are marked as active or inactive for adaptive refinement. A multigrid preconditioned CG algorithm is used to solve the linear equations for the pressure correction.

The methods reviewed in this section construct multigrid methods to allow for a fast solution of linear systems. The embedded geometry methods by Johansen and Colella [JC98], Schwartz et al. [SBCL06], and Wan and Liu [WL04] do not provide the option to handle mixed boundary conditions. The geometric multigrid method for mixed boundary conditions presented by McAdams et al. [MST10] is embedded into a CG algorithm as a preconditioner. It does not converge on its own as the discretization on different levels of resolution is not consistent. Therefore, multigrid methods building upon highly consistent mesh hierarchies are required. Chentanez and Müller build upon the variational discretization by Batty et al. [BBB07] which has been shown to be numerically less efficient than the cut-cell method by Ng et. al. [NMG09]. A multigrid method building on the cut-cell formulation has not yet been developed and investigated.

3.2.4. Interactive GPU-based methods

Although interactive fluid simulations are also possible with CPU-based approaches by significantly limiting the spatial resolution, methods targeting GPU hardware have been presented to achieve low computation times with high resolution. Therefore, the pillar *data structures and methods for massively parallel computing* is in the focus of this section. Harris [Har04] developed a GPU-based approach for two-dimensional fluid simulation using Cg shaders. He uses vorticity confinement to mitigate the numerical viscosity of semi-Lagrangian advection. The pressure correction is solved with a Jacobi method by only spending a few iterations to guarantee high and constant frame rates. Crane et al. [CLT07] presented a similar approach for three-dimensional GPU-based fluid simulations at interactive rates. Similar to Harris' method, general-purpose computing on graphics processing units (GPGPU) by means of shader programs are used to perform the numerical calculations for a fast fluid simulation. Additionally, moving solid geometry is considered by remeshing the domain and by incorporating the solid's velocity into the fluid simulation. Likewise, the solution of the pressure correction is conducted by using only a few Jacobi iterations in order to achieve interactivity. The blocky representation of solid-fluid boundaries is mitigated by correcting velocity vectors after the pressure correction by eliminating perpendicular components. Cohen et al. [CTG10] extended the previously mentioned GPU-based approaches to handle large scale simulations. They achieve this by allowing dynamic rigid body transformations of the simulation grids coupled to moving geometry like driving cars. The velocity fields are extrapolated so that the smoke and particle-based tracing of the flow can be conducted even outside the simulation domain. Chentanez and Müller [CM11b] presented an interactive version of the liquid simulation on restricted tall cell grids first introduced by Irving et al. [IGLF06]. Inspired by the shallow water equations, the height field representation is augmented with a few regular cubic cells to resolve small scale motion on the surface. Additionally, they introduce a multigrid approach to rapidly solve the pressure correction on this special type of discretization.

The GPU methods summarized in this section use a binary discretization for representing the geometry. A combination of GPU-based simulation with a geometric cut-cell discretization has not yet been analyzed. Furthermore, many GPU methods that focus on interactive fluid simulation utilize a few Jacobi iterations to compute the pressure. This does not accurately solve the corresponding linear system. Chentanez and Müller address this problem and use a GPU multigrid solver. Although their method is also applicable to standard 3D staggered grid discretizations they use it in the context of restricted tall cells. Furthermore, the approaches described here utilize standard representations for the sparse matrices in the pressure correction. A specialized data structure for implicitly representing the linear system is missing.

3.2.5. Summary

Many different approaches have been proposed for the simulation of fluid dynamics in computer graphics. Especially for Eulerian simulations based on regular grids, the fractional step method with the semi-Lagrangian advection opened up a lot of interesting options to efficiently simulate fluids for computer animation. Methods for different physical phenomena have been developed like the modeling of highly viscous fluids, liquids and multiple materials. The issue of numerical viscosity has been addressed by numerous specialized advection schemes. Adaptive discretizations and methods for sub-grid modeling have been proposed especially for free-surface simulations to better capture the liquid boundary in the numerical algorithm. Multigrid methods have been introduced to accelerate the solution of linear systems of the pressure correction especially for fluid simulations with large grids. Some approaches have been presented that focus on interactive simulations by mainly developing methods that run efficiently on GPUs.

However, open questions remain whether accuracy and computation speed can still be increased at the same time to improve the quality of fluid simulations. It has not yet been investigated how a CPU-based implementation

can be accelerated by efficiently exploiting massively parallel computing on GPU architectures. Furthermore, specialized GPU data structures to compactly represent sparse matrices in order to efficiently solve the corresponding linear systems are missing. There are multigrid methods that are based on binary discretizations and variational formulations. However, it has not been investigated yet how multigrid methods building upon an FV cut-cell discretization are performing. Furthermore, sub-grid accuracy for FV discretizations has been derived for Neumann boundary conditions. A method to achieve sub-grid accuracy for Dirichlet conditions in FV settings with simple stencils and symmetric linear systems is still missing.

The questions stated here require further investigation and are addressed in the following sections.

3.3. Interactive GPU-based fluid dynamics

In this section, a GPU-based approach that allows simulation of two-dimensional fluid flow in interactive time is presented. In order to achieve short computing times, massively parallel algorithms that can be efficiently executed on GPUs are developed. Furthermore, an interactive simulation framework is introduced that allows for changing geometry while fluid flow is simulated and visualized. In order to better represent embedded geometry on a regular grid discretization, a *geometric cut-cell approach* is devised that results in sub-grid accuracy for obstacles not aligned with the discretization. Massively parallel versions of the semi-Lagrangian approach for advection and pressure correction based on a cut-cell FD scheme are developed resulting in a significant speedup.

Additionally, all simulation building blocks developed in this section run completely on GPUs, i.e., costly transfers between GPU and CPU memory are avoided. Furthermore, as the simulation results reside in GPU memory they can be directly used for visualization. This is achieved by interpreting the simulation data as texture or GPU buffer objects and using them for rendering in false colors. This section is based on the publication [WPnSSF11]. Large parts of the text are copied verbatim with changes, additions and minor corrections. Furthermore, pictures and diagrams are reused as well.

3.3.1. Introduction

The simulation of time-dependent motion of fluids is usually a compute-intensive task. Especially high spatial or temporal resolutions require a very large amount of computational effort in every time step. A major bottleneck of the fractional step algorithm is the pressure correction, where large linear systems have to be solved. This bottleneck is addressed by developing massively parallel versions of all computational building blocks of a fluid simulation. Furthermore, a special GPU data structure for representing the matrix is proposed resulting in accelerated sparse matrix-vector products (SpMV) and faster solution of linear system. An interactive system combining fast fluid simulation, visualization and the possibility for generating and modifying geometry is presented.

3.3.2. Geometry sketching framework

In order to model different geometric scenarios that can be simulated directly, a simple set of modeling functionalities have been implemented in a two-dimensional framework. As shown in Fig. 3.11, points are generated by clicking and interpreted as geometry. These points define closed curves consisting of straight lines or free-form geometry. The latter type of geometry is realized with uniform B-Spline curves, where generated points are interpreted as B-Spline control points.

The geometry of the objects can be manipulated by selecting and modifying a set of control points. Simple operations such as translation, scaling and rotation can be performed, as shown in Fig. 3.12. Thereby, the shape's orientation, position and size can be changed interactively. By modifying subsets of control points, geometry can be changed locally as well. If geometry is modified or new shapes are created, the discretization must be updated. If the modeled geometry consists of B-Splines, the curve is sampled at multiple locations of the parameter space to generate a set of line segments. These line segments are then passed to the discretization algorithm, which is described in the following section.

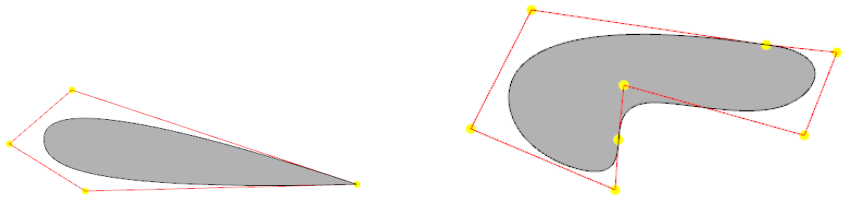


Figure 3.11.: A closed B-Spline curve representing an obstacle in the simulation can be generated by specifying control points (indicated by yellow dots).

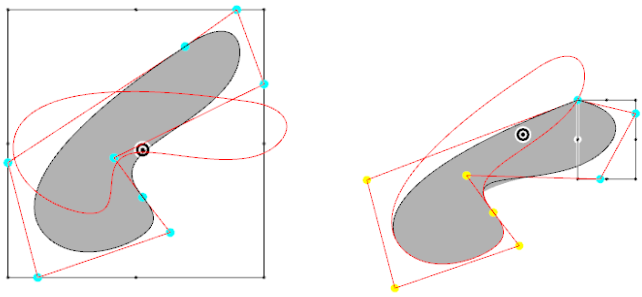


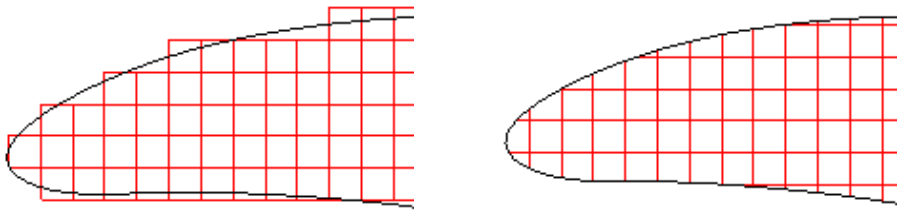
Figure 3.12.: Modifying a sketched shape: the original shape can be rotated or locally deformed.

3.3.3. Geometric cut cells

Regular grids are frequently chosen for the simulation of fluid dynamics in computer graphics, as simple FD schemes can be applied and particle tracing for semi-Lagrangian advection can be realized efficiently. To simulate flow around solid geometry, appropriate boundary conditions at fluid-solid boundaries have to be defined. A standard approach for regular grids is to mark each cell as fluid or solid in order to specify where fluid can flow. This implicitly defines the simulation domain as the union of all fluid cells. This classification can be obtained by testing if the center of each cell is inside solid geometry. Alternatively, a more conservative approach is to define a cell as being solid if at least a small fraction is covered by geometry.

However, this approach of binary discretization can only represent geometry accurately that aligns with the cell borders. As shown in Fig. 3.13 (a), geometry that is not aligned with the grid cells is not treated consistently. Hence, the resulting fluid flow near the geometry is highly inaccurate as described by Batty et al. [BBB07]. In the following, a method is developed that mitigates these artifacts using a cut-cell methodology for solid-fluid boundaries similar to Ng et al. [NMG09]. The cut cells are represented by fractionally covered interfaces between cells as depicted in Fig. 3.14 (a). These interfaces represent a percentage of how much fluid can be transported between neighboring cells in comparison to an interface without obstacles. However, in contrast to Ng et al., who represent the geometry by a level set, the discretization is explicitly determined by computing the intersections between the grid and the geometry. For each face f, which is represented by fractional indices, i.e., $i \pm \frac{1}{2} j$ or

$ij \pm \frac{1}{2}$, to indicate its position in the mesh, the face coverage A_f is computed. The representation of geometry can be given either as line segments or generated from continuous curves. In the latter case the segment length can be varied depending on the size of the mesh cells in order to guarantee the required resolution. Given a set of oriented and closed line segments, the discretization is determined as follows. For every line segment the intersecting cells must be determined. More precisely, the edges of the cells, i.e., the grid lines have to be taken into account. Therefore, the axis-aligned bounding box of a line segment is computed that encloses all potential edges. Then, for each edge within bounding box intersections with each line segment are determined. In addition, the orientation of the line segments is taken into account to determine which side belongs to the solid on fluid representation. With this information, the fractions A_f can be computed (see Fig. 3.14), which have to be incorporated in the advection and the pressure correction step.



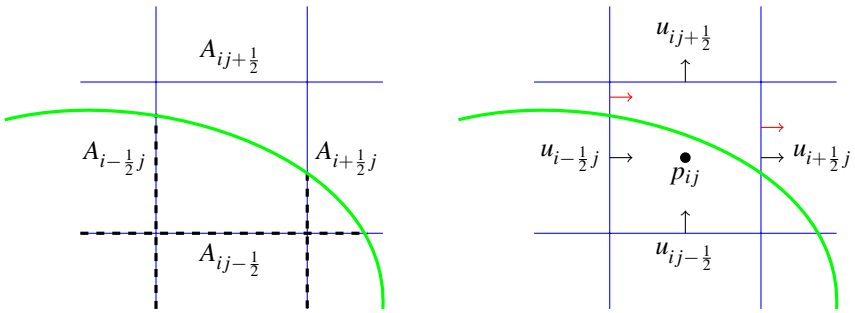
(a) (b)

Figure 3.13.: (a) The computational mesh that represents the geometry for the solver is not a good approximation in the case of binary discretization. (b) Using a cut-cell approach, the mesh can much better approximate the geometry.

While computing the advection term, solid geometry must be treated as a boundary condition, where the normal component of the velocity is constrained. Due to numerical drift in the integration step, particle trajectories may end inside an obstacle. For semi-Lagrangian advection it is important that the velocity interpolation does not induce errors due to the discretization. However, when evaluating the velocity close to the boundary, a standard bilinear interpolation would give a distorted result as velocity samples inside the geometry are zero by definition for non-moving solids. This distortion can be avoided by updating velocity samples in the solid by means of extrapolation. Therefore, velocity samples in proximity of the solid boundary are used to compute a velocity value by linear extrapolation.

A further difficulty using a cut-cell approach is that velocity samples on the regular grid discretization may be inside the solid geometry as shown in Fig. 3.14 (b). In such cases, an additional error is induced, as the starting point is not within the fluid. Even for an exact numerical integration of the particle trace following the extrapolated values, the goal position will be inside the geometry as the normal component is constrained by $\mathbf{v} \cdot \mathbf{n} = 0$. Therefore, the starting positions of the particles are modified by repositioning them to the center of the non-solid face fraction as indicated by the red arrows in Fig. 3.14 (b).

In order to obtain velocity vectors that are tangential to the embedded geometry, the pressure correction must be adapted to account for the cut cells. First, the pressure correction is computed by solving the Poisson equation (3.32) as derived in Section 3.1.2.3 adapted to a two-dimensional simulation. The divergence of the velocity is



(a) (b)

Figure 3.14.: (a) In the proposed cut-cell approach, interface fractions are determined that represent the area over which fluid can be exchanged between adjacent cells. As an example, the interface at $A_{i-\frac{1}{2}j}$ allows less fluid to be exchanged in comparison to interface $A_{i+\frac{1}{2}j}$. (b) When using cut cells, velocity samples such as $u_{i-\frac{1}{2}j}$ are positioned within the solid geometry on the original mesh. In order to reduce the numerical error of particle tracing in the advection step, the start point is placed at the center of each interface as shown with the red arrows.

computed by modifying Eq. (3.35) to additionally account for cut cells by incorporating the face fractions

$$(\nabla \cdot \mathbf{v})_{ij} = \frac{A_{i+\frac{1}{2}j}u_{i+\frac{1}{2}j} - A_{i-\frac{1}{2}j}u_{i-\frac{1}{2}j} + A_{ij+\frac{1}{2}}v_{ij+\frac{1}{2}} - A_{ij-\frac{1}{2}}v_{ij-\frac{1}{2}}}{\Delta x}. \tag{3.43}$$

Similarly, the coefficients in the linear system for the discretized Poisson equation given in Eq. (3.38) must be modified to account for the cut cells. Then, the left hand side of Eq. (3.38) is changed to

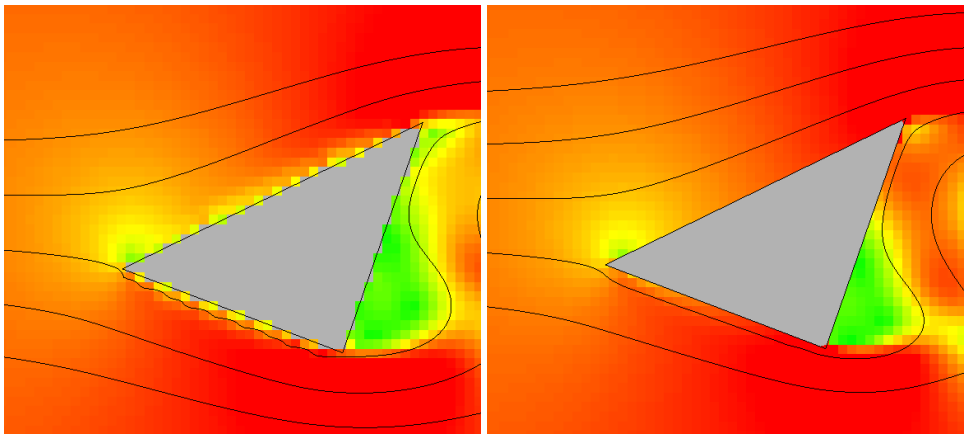
$$\frac{-A_{i+\frac{1}{2}j}p_{i+1j} - A_{ij+\frac{1}{2}}p_{ij+1} + A_{ij}p_{ij} - A_{i-\frac{1}{2}j}p_{i-1j} - A_{ij-\frac{1}{2}}p_{ij-1}}{\Delta x^2}, \tag{3.44}$$

where $A_{ij} = A_{i+\frac{1}{2}j} + A_{i-\frac{1}{2}j} + A_{ij+\frac{1}{2}} + A_{ij-\frac{1}{2}}$ is the sum of fractions adjacent to cell C_{ij}. In the case of a pure solid cell with no fluid interfaces, all adjacent face fractions are zero. Then the diagonal A_{ij} is set to 1 in order to avoid a zero line in the matrix and to avoid a singular linear system, which is equivalent to removing the corresponding cell.

The pressure update is unmodified and the updated velocities are computed according to Eq. (3.36). Employing pressure correction with cut cells, i.e., adapting the computation of divergence and the matrix of the linear system, results in a velocity field that respects the embedded geometry as depicted in Fig. 3.15 (b). By using the cut-cell approach, the flow is automatically aligned along the geometry due to the pressure correction.

3.3.4. GPU-based fluid simulation

In the following, the algorithm and the massively parallel versions of the building blocks are described that efficiently compute two-dimensional fluid flow on GPUs. The parallelization of the algorithmic building blocks is realized using NVIDIA CUDA [NVI15] as outlined in Section 2.5.2. The resolution of the simulation domain is chosen to be a power of two, as this yields uniform and aligned access pattern to GPU memory. Alternatively,



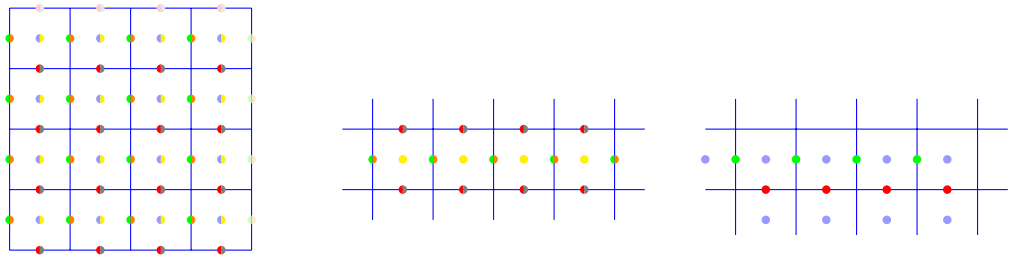
(a) (b)

Figure 3.15.: The flow around an obstacle using a binary (a) and a cut-cell discretization (b). The colors in the pictures represent the magnitude of velocity from low in green to high in red. The non-smooth stream lines and the non-smooth velocity distribution in (a) near the obstacle are an effect of the binary discretization. (b) A cut-cell discretization that results in a smooth velocity field.

the memory can be padded for other resolutions. As emphasized in Section 2.5.2, good GPU performance is achieved when memory bandwidth is exploited optimally. Therefore, coalesced memory access has to be ensured to minimize memory transactions.

For fluid simulations on staggered grids, several arrays are required to store the state of the system. These include arrays for the velocity, pressure and divergence as well as the discretization given by the face fractions. The data is associated with the respective cells of the discretization and stored linearly in memory, where the size corresponds to the grid size of $n_x n_y$. However, the velocity components and face fractions are not directly associated with cells, but rather to faces, which is indicated by fractional indices as shown in Fig. 3.14 (b). Therefore, the convention is that values with the lower index are associated with the cell C_{ij}. For example, $u_{i-\frac{1}{2}j}$ and $v_{ij-\frac{1}{2}}$ are stored in the u and v array at the position corresponding to the indices i and j. In order to ensure aligned memory access, the sizes of the velocity and face fraction arrays are also chosen to be $n_x n_y$ as shown in Fig. 3.16 (a). However, special treatment is required when accessing values at the upper and right boundary, where the specified boundary conditions must be used instead of accessing elements in the array.

The kernels described in the following are designed with the goal that groups of threads operate on adjacent memory. A regular grid discretization directly allows for such memory access patterns. Most kernels compute an update for each cell and a natural launch configuration is linear indexing, i.e., consecutive threads process consecutive items in memory. For example, the kernel for computing the divergence in Eq. (3.43) calculates the cell-centered values $(\nabla \cdot \mathbf{v})_{ij}$ for each thread. Furthermore, the kernel has to read all velocity and face fraction samples that are located at adjacent faces. The access patterns to the corresponding values are depicted in Fig. 3.16 (b), where the different data sets involved are highlighted by color: horizontal velocity components $u_{i-\frac{1}{2}j}$ and face fractions $A_{i-\frac{1}{2}j}$ in green and orange, vertical velocity components $v_{ij-\frac{1}{2}}$ and face fractions $A_{ij-\frac{1}{2}}$ in red and gray, and divergence samples in yellow. The resulting memory transactions are aligned. However, the horizontal velocity samples and face fractions require an additional load operation as one more value than the



(a) (b) (c)

Figure 3.16.: (a) Example for a 4×4 grid discretization with horizontal velocity (green), vertical velocity (red), horizontal face fractions (orange), vertical face fractions (gray), pressure (blue) and divergence (yellow) samples. Velocity and face fraction samples require larger arrays than cell-centered samples. In order to enable aligned memory access, samples in lighter color at the top and right boundary are not explicitly stored. (b) The divergence computation requires to access adjacent velocity and face fraction samples. With an exemplary warp a size of four, the memory accesses are aligned and can be coalesced. However, the horizontal velocity and face fraction samples require an additional load operation. (c) The pressure update computes new values for the horizontal and vertical velocity samples. It requires to access the pressure samples adjacent to the respective velocities. Again, with an exemplary warp a size of four, the memory accesses are aligned and can be coalesced. However, horizontal pressure samples require an additional load operation.

number of cells is necessary for the computation. The kernel computing the pressure update (Eq. (3.36)) exhibits a different access pattern, as samples at other locations are involved. For each index i and j the corresponding thread updates the velocities $u_{i-\frac{1}{2}j}$ and $v_{ij-\frac{1}{2}}$ as depicted in Fig. 3.16 (c). Therefore, the pressure samples p_{ij}, p_{i-1j} and p_{ij-1} are loaded to determine the pressure gradients, which are then used to update the velocity samples. The type of memory transactions is similar as the load and write operation for the velocity samples are aligned and can be coalesced. Similarly, pressure samples require one more load operation than the number of cells.

3.3.4.1. Advection

The advection step is conceptually similar to the algorithm described in Section 3.1.2.2. For the GPU approach, advection is computed in parallel by letting each thread compute the two velocity samples associated with one cell C_{ij}, i.e., $u_{i-\frac{1}{2}j}$ and $v_{ij-\frac{1}{2}}$. Particle tracing is conducted for each of the samples using a second order Runge-Kutta integration. The evaluation of the velocities for numerical integration and final evaluation are determined by bilinear interpolation. The initial memory read and the final write operation can be executed in a coalesced memory transaction as adjacent threads access adjacent regions of memory. However, the read operations for the intermediate steps and at the final position to determine the velocity for particle tracing depend on the state of the velocity field. Therefore, it is difficult to ensure regular access patterns as the particles might travel into different directions. However, a reasonable assumption is that the velocity field does not vary significantly for adjacent particles and access patterns are therefore similar.

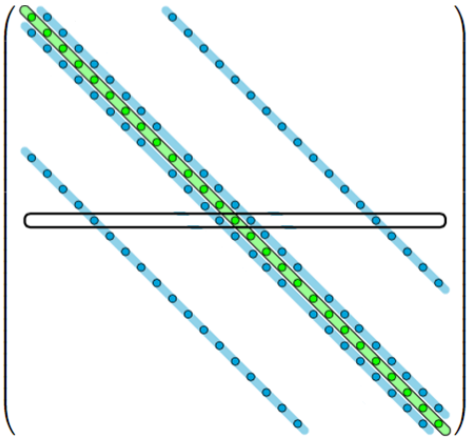


Figure 3.17.: A two-dimensional FD discretization results in the so-called five-point stencil, where there are up to five non-zero entries in a matrix row, which can be represented by five arrays shown as blue and green bands.

3.3.4.2. Implicit matrix data structure for pressure correction

In order to compute the pressure, the corresponding linear system is solved using a PCG algorithm. The algorithm consists of vector additions, scalar multiplications, dot products and SpMV operations, where the latter is the most time consuming operation. As the matrix is large and sparse, only non-zero entries are stored and used for SpMV. For a two-dimensional simulation there are up to five non-zero entries per row. A standard approach to store non-zero entries is to use the *five-point stencil data structure* with the arrays $A^0,\ldots,A^4$ that represent the individual diagonals in the matrix A. Each cell C_{ij} is associated the r'th row given by $r = i + jn_x$. The r'th entry in each array $A^0[r],\ldots,A^4[r]$ represents the non-zero entry in the r'th row of the matrix as shown in Fig. 3.17. The columns are implicitly given by the index of adjacent cells $C_{i\pm1\,j}$ or $C_{ij\pm1}$. These values are used to compute one row of the matrix-vector multiplication

$$y_r = \sum_{k=0}^{4} A^k[r]x_l, \tag{3.45}$$

where l is the index representing the column.

In order to minimize the number of transfers required for a matrix-vector multiplication, an implicit representation of the matrix based on the face fractions $A_{i\pm\frac{1}{2}j}$ and $A_{ij\pm\frac{1}{2}}$ is proposed. The entries of a matrix row r can be determined by evaluating the face fractions as specified in Eq. (3.44). In order to reconstruct the matrix entries $A^0[r],\ldots,A^4[r]$ from $A_{i\pm\frac{1}{2}j}$ and $A_{ij\pm\frac{1}{2}}$ in the kernel, the following computations are required

$$\begin{aligned}
A^0[r] &= A_{ij-\frac{1}{2}} \\
A^1[r] &= A_{i-\frac{1}{2}j} \\
A^2[r] &= A_{i-\frac{1}{2}j} + A_{ij-\frac{1}{2}} + A_{i+\frac{1}{2}j} + A_{i+\frac{1}{2}j} \\
A^3[r] &= A_{i+\frac{1}{2}j} \\
A^4[r] &= A_{ij+\frac{1}{2}},
\end{aligned}$$

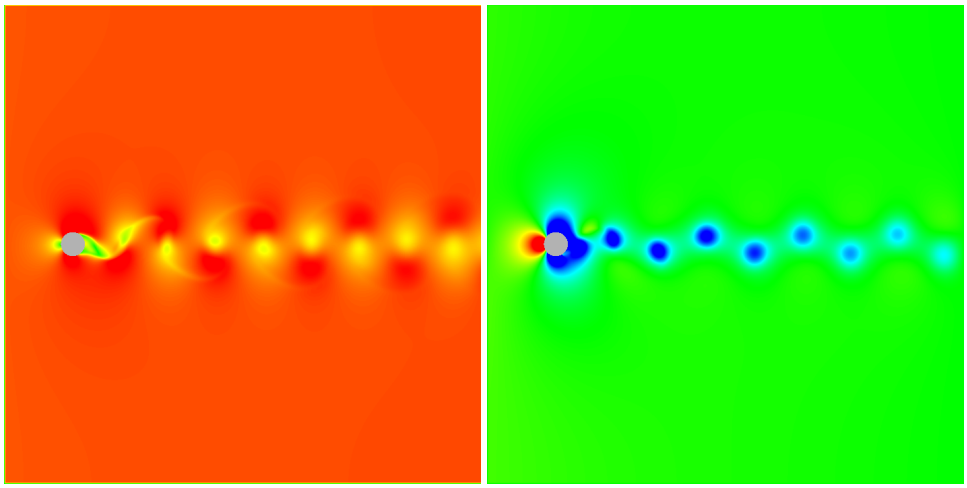


Figure 3.18.: The test scenario used for the performance evaluation. A circular obstacle is subject to flow from left to the right and develops the so-called Kármán vortex street. The colors in the left picture represent the magnitude of velocity from low in green to high in red, whereas the right picture shows the pressure distribution from low in blue to high in red color.

where $r = i + jn_x$. Furthermore, the GPU cache enables to reuse some of the entries, as the linear system is symmetric and, e.g., $A^1[r] = A^3[r+1]$. The value on the diagonal A_{ij} is determined by accumulating all adjacent face contributions as described in Section 3.3.3. Therefore, the amount of data is reduced and the number of arrays passed to the kernel is decreased from five to two. The required data transfers can be coalesced as overlapping data in memory is implicitly reused.

3.3.5. Results

In this section, the GPU-based solver together with the two-dimensional simulation framework for geometry modifications is analyzed. The performance of the GPU-based solver is evaluated by varying the spatial resolution of the meshes, by comparing it to a CPU implementation and by analyzing the effect of the implicit matrix data structure.

3.3.5.1. Performance analysis

In the following, the performance of the two-dimensional GPU-based solver is analyzed. All computations were performed on an Intel Core i3 with 3.2 GHz using only one core with an NVIDIA GeForce GTX 460 as GPU. As a test scenario, a two-dimensional simulation with a circular obstacle in a wind tunnel was selected, producing the typical Kármán vortex street after a certain amount of time (see Fig. 3.18). For varying spatial resolutions, the time step was adjusted such that the *CFL number* is constant w.r.t. the inlet velocity. The CFL number is a dimensionless quantity, which represents the number of cells a virtual particle passes in one time step. In many cases it is used to guarantee the stability of numerical algorithms, but a low CFL number also ensures physical accuracy w.r.t. time. Even with unconditionally stable algorithms, excessively large time steps may significantly reduce the accuracy and sometimes even produce non-physical behavior. Therefore, a comparatively low CFL

Resolution	t_{adv}	t_{div}	t_{sol}	t_p
32^2	0.21	0.06	6.85	0.11
64^2	0.26	0.09	13.02	0.05
128^2	0.31	0.08	31.01	0.06
256^2	0.38	0.10	84.48	0.07
512^2	0.56	0.14	491.70	0.13
1024^2	1.31	0.41	3526.54	0.30
2048^2	4.52	1.36	37264.35	1.00

Table 3.1.: Average computing times in (ms) for one time step using the GPU solver split into the individual parts: Advection t_{adv}, computation of divergence t_{div}, solution of the linear system t_{sol} and pressure correction t_p.

Resolution	32^2	64^2	128^2	256^2	512^2	1024^2	2048^2
iterations	103	189	377	640	1396	2705	5411

Table 3.2.: The average number of required CG iterations w.r.t. the number of degrees of freedom to achieve a relative residual improvement of 10^{-5} are shown.

number of 0.5 was chosen. However, using a constant CFL number has implications on the runtime w.r.t. cell size. Doubling the resolution requires twice the number of time steps to simulate the same physical time.

As a first test, 100 time steps of the scenario described above were simulated with varying mesh resolutions. The CG solver was configured to iteratively solve the linear system until the residual norm is reduced by a factor of 10^{-5}. This is done by determining the residual vector $\mathbf{r}_i = \mathbf{b} - \mathbf{A}\mathbf{x}_i$ in iteration i and evaluating if $\frac{\|\mathbf{r}_i\|}{\|\mathbf{r}_0\|} < 10^{-5}$.

The measured and averaged times are listed in Table 3.1. All parts of the algorithm except the solution of the linear system show a linear behavior w.r.t. the number of degrees of freedom $n = n_x n_y$ as shown in Fig. 3.19. In contrast, the time required for solving the linear system does not show linear scaling. This is due to the fact that achieving convergence for a larger linear system requires a larger number of iterations. The runtime of the CG solver is dominated by SpMV, as this is the most expensive part. It requires $O(m)$ operations, where m is the number of non-zero elements in the matrix. As there are no more than five non-zero entries per row, the matrix-vector multiplication has a complexity of $O(n)$. Shewchuk [She94] noted that the number of iterations i required for convergence depends on the condition number κ of the matrix with $i \in O(\sqrt{\kappa})$. For two-dimensional FD schemes as adopted here for the GPU solver, Shewchuk estimates a condition number of $\kappa \in O(n)$, so $i \in O(\sqrt{n})$ iterations are required for convergence. Overall, the CG algorithm is of order $O(n^{3/2})$ as $O(n^{1/2})$ iterations are needed and one iteration requires $O(n)$ operations for the matrix-vector multiplication. Table 3.2 lists the average number of CG iterations required to achieve a relative residual of 10^{-5} for various mesh resolutions. The number of iterations matches Shewchuk's estimation as they grow sub-linearly w.r.t. the dimension. From the timings, it is apparent that for high resolutions the solution of the linear system and therefore the whole simulation algorithm becomes prohibitively expensive. However, even when accurately solving the linear system for the pressure correction, a resolution of 128^2 can be simulated at interactive rates as nearly 20 updates per second are possible. Additionally, a resolution of 256^2 is close to achieving ten updates per second.

One option for simulating with high resolutions at moderate computing times is to limit the number of CG iterations. This would result in a linear scale up w.r.t. the degrees of freedom. Harris et al. [Har04] and Crane et al. [CLT07] propose a similar approach and use a constant number of Jacobi iterations to solve the linear system. With $40-80$ iterations they achieve visual plausibility for their simulation scenarios. However, using

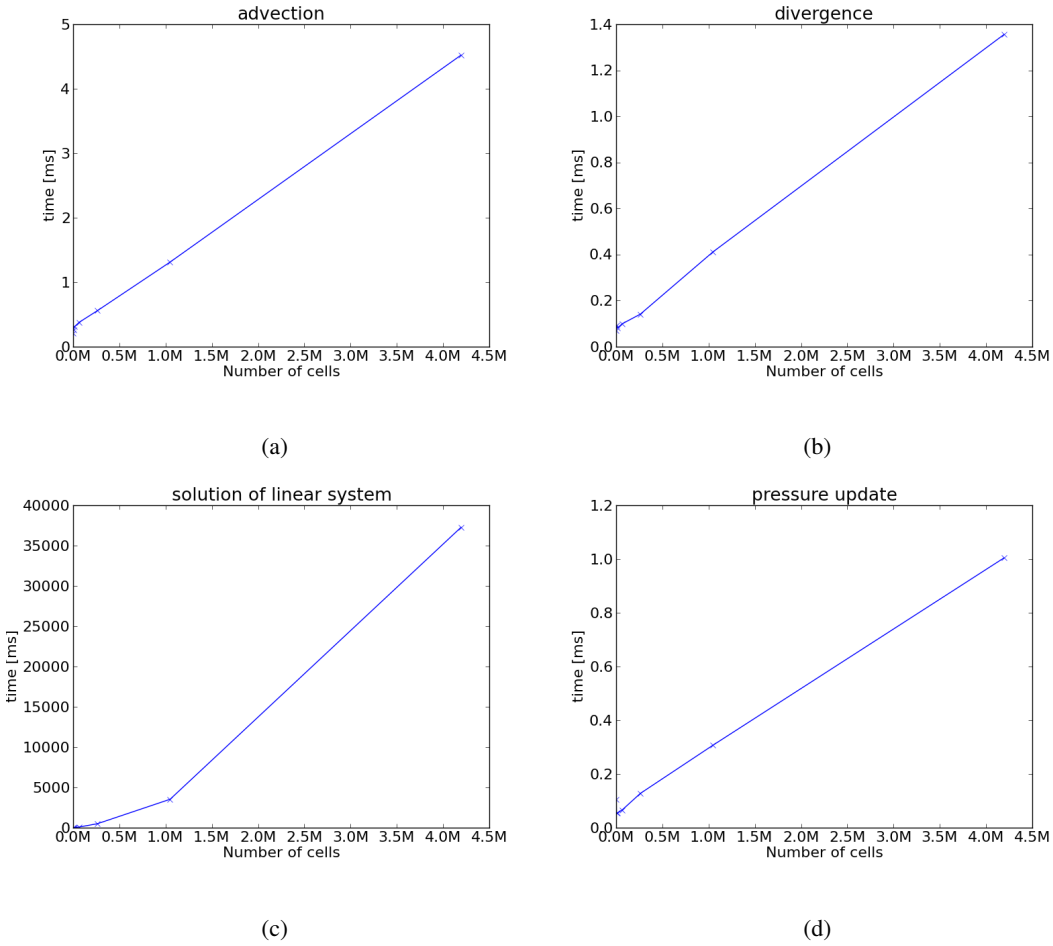


Figure 3.19.: Average computing times in (ms) over the number of cells. (a) Advection, (b) computation of divergence, (c) solution of the linear system and (d) pressure correction. All steps except the solution of the linear system scale linearly w.r.t. number of cells.

Resolution	32^2	64^2	128^2	256^2	512^2	1024^2
CPU time (ms)	4.75	19.29	79.78	321.87	1446.91	6196.56
GPU time (ms)	29.96	30.62	35.00	58.03	128.21	439.49
Acceleration	0.16	0.63	2.28	5.55	11.29	14.10

Table 3.3.: Average time (in ms) to compute one time step using the CPU and the GPU implementation. The resulting acceleration reaches a factor of up to 14.

Resolution	32^2	64^2	128^2	256^2	512^2	1024^2
Five-point stencil data structure	0.39	0.41	0.73	2.73	10.80	42.59
Implicit matrix data structure	0.40	0.44	0.74	3.15	12.68	51.16
Acceleration	1.03	1.07	1.01	1.15	1.17	1.20

Table 3.4.: Timings in (ms) to execute 100 SpMV operations using the standard five-point stencil data structure and the implicit matrix data structure. Using the implicit matrix data structure results in a speedup of up to 20%.

Jacobi iterations is not an efficient strategy for accurately solving the linear system. Even with 1000 iterations the residual is only reduced by 63% on average in the 512^2 scenario. With a CG algorithm, this reduction is achieved with only 65 iterations. The time required to perform this fixed number of iterations is 122 ms for the Jacobi and 43 ms for the CG algorithm resulting in an acceleration of approximately three.

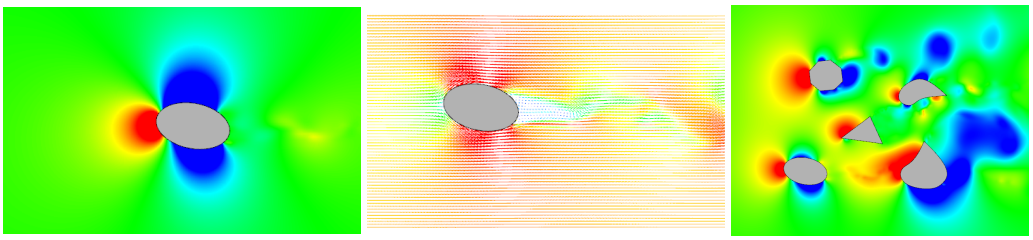
When using a very small number of iterations for the CG algorithm, i.e., below 15 iterations, oscillations in the pressure field are generated. However, this non-physical behavior does not affect the velocity field. For the 1024^2 scenario ten CG iterations still result in simulations that develop the vortex street. By imposing this iteration limit, the solution of the linear system is computed in approximately 30 ms and the overall time step only requires 36 ms. Therefore, a visual plausible simulation with more than one million degrees of freedom can be conducted with more than 25 updates per second.

In the following, the performance of the GPU solver and a CPU version is compared. The performance measurements are evaluated for a single-core CPU implementation. Due to a different order of operations in the implementations, the number of iterations until convergence may vary because of floating point precision issues. Therefore, a fixed number of 200 CG iterations was chosen for a fair runtime comparison. Table 3.3 lists the average times required for one time step for both solvers. As listed in the table, the GPU implementation accelerates the fluid simulation by a factor of up to 14 in the simulation scenarios.

Now, the effect of the implicit matrix data structure is analyzed. Therefore, 100 matrix-vector multiplications are conducted with the five-point stencil data structure and compared to the implicit representation. Table 3.4 lists the computation times for different resolutions. Utilizing the implicit matrix data structure results in a speedup of up to 20%. This is due to the reduced amount of data that must be transferred to the computation kernels.

3.3.5.2. Shape sketching and geometry modification

Fig. 3.20 shows an example of the sketching functionality embedded in a simulation framework. An ellipsoid geometry is sketched and put into a wind tunnel as an obstacle. Fig. 3.20 (a) and (b) show a false coloring of the resulting pressure field and the velocity vector field. The simultaneous sketching, simulation and visualization functionality is shown in Fig. 3.20 (c). There, multiple parts are sketched into the simulation domain while the simulation is running. The newly sketched shapes are immediately integrated into the current simulation by



(a) (b) (c)

Figure 3.20.: (a) The fluid flow around a deformed cylinder with the pressure field as color from low in blue to high in red and (b) velocity as vector field with colors representing the magnitude from low in green to high in red. (c) Insertion of multiple parts with a direct simulation and visualization of the pressure.

means of the automatic discretization process. Updating the cut-cell discretization only takes a few milliseconds. Even for a resolution of 1024^2 grid cells and many geometric entities the discretization time is clearly below 25ms.

3.3.6. Summary

In this section, an interactive solver for two-dimensional fluid flow has been presented with the following contributions. Based on a regular grid discretization, a massively parallel version of the fractional step method was developed to exploit the high potential of GPUs. In order to reduce the amount of data to be transferred to the GPU kernels, an *implicit matrix data structure* was designed for efficiently solving the linear system required in the pressure correction. The fast computation times enabled a framework that allows for geometry modifications and direct visualization while the simulation is running. Furthermore, the *geometric cut-cell method* was introduced to improve the representation of embedded geometry on regular grids by directly intersecting the geometry with the grid. The runtime behavior w.r.t. varying discretizations was examined, different sparse matrix data structures were tested and a comparison to a CPU implementation was conducted.

It has been identified that the solution of the linear system is the bottleneck of the simulation algorithm. In Section 3.4, a multigrid method based on the geometric cut-cell discretization is developed that solves the equations more quickly even for a very large number of degrees of freedom. Additionally, the approach described here is extended to three-dimensional flow and to free-surface simulations.

3.4. The cut-cell geometric multigrid method

In this section, a novel multigrid scheme based on an FV cut-cell formulation on regular staggered grids is presented that generates compatible systems of linear equations on all levels of the multigrid hierarchy. This cut-cell multigrid method is a geometrically motivated formulation, derived from an FV approach and exhibits an improved rate of convergence compared to previous methods. Existing fluid solvers with voxelized domains can directly benefit from this approach by only modifying the representation of the non-fluid domain. The necessary building blocks are fully parallelizable and can therefore benefit from multi- and many-core architectures. This section is based on the publication [WMRSF15]. Large parts of the text are copied verbatim with minor additions and corrections. Furthermore, pictures and diagrams are reused as well.

3.4.1. Introduction

Fluid simulations resolving small and large scale phenomena require a large number of degrees of freedom. Many applications use structured, orthogonal and equidistant staggered grids, first introduced to the computer graphics community by Foster and Metaxas [FM96], with semi-Lagrangian advection (see [Sta99]). At the core of this well-known and widespread method, the pressure projection step requires the solution of a large system of linear equations. This is computationally intensive and usually takes most of each simulation step's time. Many authors (see, e.g., [Bri08]) use an ICCG method, which is ill-suited for parallel implementations as back substitution is difficult to parallelize. More recent works use multigrid methods for pressure projection as a preconditioner [MST10], using a variational discretization [CM11a, CM11b] or using an adaptive FE discretization for very large simulations [FWD14].

In the following, a geometric multigrid scheme based on the geometric cut-cell method described in Section 3.3 is constructed that produces consistent mesh hierarchies, simple coarsening schemes and symmetric matrices on every level. This in turn leads to improved convergence for Neumann boundary conditions. Additionally, the handling of Dirichlet boundary conditions in a cut-cell FV approach is addressed, which are only first order accurate but maintain symmetry in the resulting linear system. Cells on coarser levels are not duplicated to maintain memory alignment especially for GPU implementations. The cut-cell multigrid method can either be used as a standalone multigrid solver or embedded in a CG algorithm as a preconditioner.

Existing fluid solvers based on voxelized regular grids can directly benefit by slightly modifying the solid-fluid boundary representation. In comparison to state-of-the-art algorithms, the cut-cell multigrid method results in a

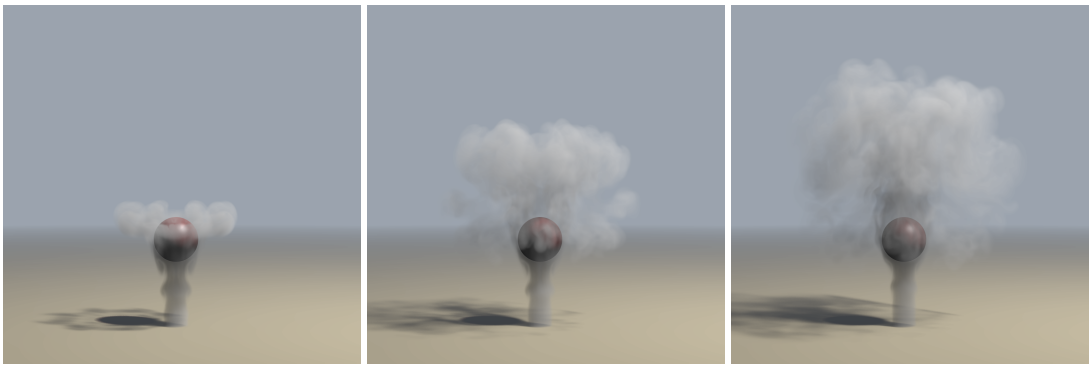


Figure 3.21.: Three frames from a simulation of smoke flowing past a sphere on a $256 \times 512 \times 256$ grid.

speedup of up to three for a 256^3 grid. In addition, a GPU implementation of the multigrid solver is shown which allows for fluid simulations with up to four million degrees of freedom at interactive rates, i.e., with simulation times below 100 ms on current hardware.

3.4.2. Discretization

To simulate incompressible fluids with constant density ρ it is necessary to compute a divergence-free velocity field $\mathbf{v}$. As described in Section 3.3, the incompressibility constraint can be enforced with a pressure projection step by determining a pressure field p which satisfies the Poisson equation

$$\nabla \cdot \nabla p = \frac{\rho}{\Delta t} \nabla \cdot \mathbf{v}, \tag{3.46}$$

on a domain Ω subject to boundary conditions on Γ, where Δt is the time step. The boundary value problem is discretized on a structured, orthogonal and equidistant staggered grid. This setting is depicted in Fig. 3.22 and is described in detail in the textbook by Bridson [Bri08] and in Section 3.1.

For irregular domains, a simple approach is to use a voxelized domain, i.e., to mark each cell either as interior, solid or air to model the fluid, walls or free surfaces. An FD discretization of Eq. (3.46) then yields a system of linear equations $\mathbf{Lx} = \mathbf{b}$, where the sparse matrix $\mathbf{L}$ consists of seven entries per row due to the regular mesh topology (five per row in 2D). The entries in this matrix depend on the state of the adjacent cells. A matrix-free approach can be used (see, e.g., [MST10]), as SpMV and other operations can be deduced from the local configuration (the state of the neighboring cells).

In comparison to Section 3.3, an alternative discretization of the Poisson equation is used by applying an FV scheme similar to Ng et al. [NMG09]. Therefore, Eq. (3.46) is transformed into the weak form by integrating both sides and applying the divergence theorem. For the left hand side

$$\iiint\limits_{C_{ijk} \cap \Omega} \nabla \cdot \nabla p \, \mathrm{d}V = \oiint\limits_{\partial(C_{ijk} \cap \Omega)} \mathbf{n} \cdot \nabla p \, \mathrm{d}A \tag{3.47}$$

is obtained, where C_{ijk} is a voxel grid cell. $\mathrm{d}V$ and $\mathrm{d}A$ are infinitesimal volume and area elements, respectively. Similarly,

$$\frac{\rho}{\Delta t} \iiint\limits_{C_{ijk} \cap \Omega} \nabla \cdot \mathbf{v} \, \mathrm{d}V = \frac{\rho}{\Delta t} \oiint\limits_{\partial(C_{ijk} \cap \Omega)} \mathbf{n} \cdot \mathbf{v} \, \mathrm{d}A \tag{3.48}$$

is obtained for the right-hand side.

By introducing cell and face averages (denoted by an overline), Eqns. (3.47) and (3.48) can be written as discrete sums

$$V_{ijk} \overline{\nabla \cdot \nabla p} = \sum_f A_f \overline{\mathbf{n}_f \cdot \nabla p} \tag{3.49}$$

and

$$\frac{\rho}{\Delta t} V_{ijk} \overline{\nabla \cdot \mathbf{v}} = \frac{\rho}{\Delta t} \sum_f A_f \overline{\mathbf{n}_f \cdot \mathbf{v}}, \tag{3.50}$$

where f are the faces, V_{ijk} is the volume of cell C_{ijk} and A_f is the area of face f.

For regular faces which are part of the cell surface ∂C_{ijk} but not part of the domain boundary Γ (Γ includes free surfaces) the gradient term $\overline{\mathbf{n}_f \cdot \nabla p}$ can be approximated by a FD

$$\overline{\mathbf{n}_f \cdot \nabla p} \approx \frac{p_F - p_{ijk}}{\Delta x}, \tag{3.51}$$

as the cell centers are on a regular, orthogonal grid and aligned with the face normals. Δx is the cell spacing and p_F is the pressure value of the cell on the opposing side of f. The $\overline{\mathbf{n}_f \cdot \mathbf{v}}$ term in Eq. (3.50) corresponds to the velocity value $\pm u_f$ stored on the cell face (the sign depends on the reference normal and the face normal).

For a regular cell lying completely inside the domain, this results in the volume-scaled discretization of Eq. (3.46)

$$\Delta x \left(\ldots - p_{i-1jk} + 6p_{ijk} - p_{i+1jk} \ldots \right) = -\tfrac{\rho}{\Delta t} \Delta x^2 \left(u_{i+\frac{1}{2}jk} - u_{i-\frac{1}{2}jk} + v_{ij+\frac{1}{2}k} - \ldots \right), \tag{3.52}$$

as all face areas A_f equal Δx^2 (both sides are negated to create a positive definite system of linear equations). For such cells, the discretization differs from the commonly used FD discretization only by a scale factor of $V_{ijk} = \Delta x^3$. For cell faces partially cut by the irregular domain boundary, A_f varies between 0 and Δx^2. As the main diagonal equals the negated sum of the off-diagonal elements and the face areas are equal for all adjacent cells, the resulting matrix remains positive definite and diagonally dominant in all cases.

For Neumann boundary conditions (e.g., walls, fixed velocity inlets), the pressure gradient along the face normal is defined as

$$\frac{\partial p}{\partial \mathbf{n}_f} = a. \tag{3.53}$$

Therefore, the corresponding term

$$A_f \overline{\mathbf{n}_f \cdot \nabla p} = A_f a \tag{3.54}$$

can be moved to the right-hand side and does not affect the resulting matrix.

McAdams et al. [MST10] use the common approach where Dirichlet boundaries (e.g., free surfaces, fixed pressure inlets) are enforced at ghost cells which lie beyond the actual boundary instead of at the boundary. Ng et al. [NMG09] do not consider Dirichlet boundaries at all. Chentanez and Müller [CM11b] achieve this using the ghost cell method by extrapolating a negative pressure outside of the domain, ensuring that $p = 0$ at the boundary. In the method presented here, Dirichlet boundary conditions are applied at the corresponding geometric boundary. Therefore, an FD approximation

$$A_f \overline{\mathbf{n}_f \cdot \nabla p} \approx 2A_f \frac{b - p_{ijk}}{\Delta x} \tag{3.55}$$

for grid-aligned boundaries is used, where b is the boundary pressure, as the distance to the cell boundary is $\frac{\Delta x}{2}$, which is equivalent to applying the GFM proposed by Enright et al. [ENGF03]. The known component $2A_f \frac{b}{\Delta x}$ is moved to the right-hand side and $2A_f \frac{1}{\Delta x}$ is added to the main diagonal of the discretized matrix.

For non-aligned Dirichlet boundaries ∇p is approximated as

$$\nabla p \approx \operatorname{sgn}(\mathbf{n}) 2 \frac{b - p_{ijk}}{\Delta x}, \tag{3.56}$$

where $\mathbf{n}$ is the normal of the boundary where ∇p is evaluated and sgn is the component-wise sign function, i.e., the grid-aligned value with the same sign of the normal as the current component is used. Then

$$\begin{aligned} \oiint_{\partial(C_{ijk} \cap \Omega)} \mathbf{n} \cdot \nabla p \, \mathrm{d}A &\approx 2 \frac{b - p_{ijk}}{\Delta x} \sum_{k \in x,y,z} \oiint_{\partial(C_{ijk} \cap \Omega)} |n_k| \, \mathrm{d}A \\ &= 2 \frac{b - p_{ijk}}{\Delta x} \sum_{k \in x,y,z} \oiint_{\partial(C_{ijk} \cap \Omega)} |\cos \beta_k| \, \mathrm{d}A = 2A_d^{proj} \frac{b - p_{ijk}}{\Delta x}, \end{aligned} \tag{3.57}$$

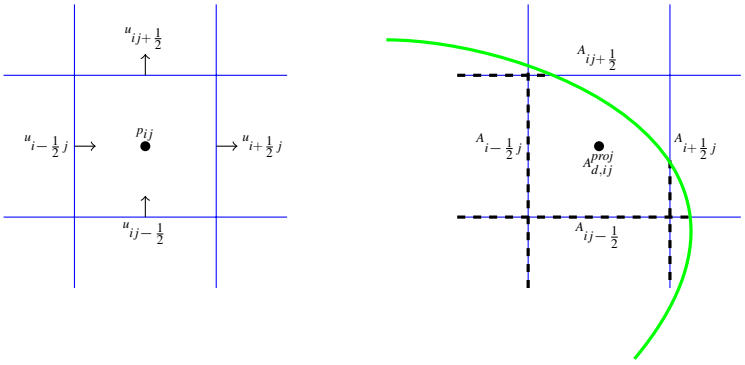


Figure 3.22.: Two-dimensional staggered grid and storage locations of discretized areas. The dashed lines represent areas inside obstacles and the green line corresponds to the domain boundary Γ.

where β_k is the angle between the normal and the unit vector $\mathbf{e}_k$ along axis k and A_d^{proj} is the unsigned, projected area of the Dirichlet boundary within the cell. This discretization is consistent with the restriction and prolongation operators chosen in the multigrid algorithm and yields better convergence than always moving the Dirichlet boundary to the closest cell boundary on all levels. Effectively, non-aligned boundaries are approximated as aligned boundaries with partial coverage of cells faces.

Combining all of the above, the matrix is constructed as

$$\begin{aligned}
\mathbf{L}_{r_{ijk},r_{i\pm1jk}} &= -\frac{A_{i\pm\frac{1}{2}jk}}{\Delta x} \\
\mathbf{L}_{r_{ijk},r_{ij\pm1k}} &= -\frac{A_{ij\pm\frac{1}{2}k}}{\Delta x} \\
\mathbf{L}_{r_{ijk},r_{ijk\pm1}} &= -\frac{A_{ijk\pm\frac{1}{2}}}{\Delta x} \\
\mathbf{L}_{r_{ijk},r_{ijk}} &= \sum_{o\in\pm\frac{1}{2}} \frac{A_{i+ojk}}{\Delta x} + \sum_{o\in\pm\frac{1}{2}} \frac{A_{ij+ok}}{\Delta x} + \sum_{o\in\pm\frac{1}{2}} \frac{A_{ijk+o}}{\Delta x} + 2\frac{A_d^{proj}}{\Delta x}
\end{aligned} \tag{3.58}$$

and the right-hand side as

$$\begin{aligned}
\mathbf{b}_{r_{ijk}} = -\frac{\rho}{\Delta t} \quad &\Big(A_{i+\frac{1}{2}jk}u_{i+\frac{1}{2}jk} - A_{i-\frac{1}{2}jk}u_{i-\frac{1}{2}jk} + A_{ij+\frac{1}{2}k}u_{ij+\frac{1}{2}k} - A_{ij-\frac{1}{2}k}u_{ij-\frac{1}{2}k} + \\
&A_{ijk+\frac{1}{2}}u_{ijk+\frac{1}{2}} - A_{ijk-\frac{1}{2}}u_{ijk-\frac{1}{2}} + A_d\overline{\mathbf{n}_d\cdot\mathbf{v}_d}\Big) + A_n a + 2A_d^{proj}\frac{b}{\Delta x},
\end{aligned} \tag{3.59}$$

where r_{ijk} is the row or column corresponding to C_{ijk} and A_n is the total Neumann surface area. The Dirichlet divergence term $A_d\overline{\mathbf{n}_d\cdot\mathbf{v}_d}$ is either approximated using an extrapolated velocity or computed in the same fashion as the other divergence terms for grid-aligned boundaries.

This alternative discretization can be interpreted as follows: The areas A_f between the faces implicitly model the boundaries of the fluid. The values $A_f \in [0,\Delta x^2]$ represent the fraction of area over which fluid can be exchanged between adjacent cells (see Fig. 3.22 for a two-dimensional example). The resulting discretization yields cut cells, where the FV approach takes care of the irregular cells at boundaries. This allows to resolve the boundary of the geometry more accurately, especially when multiple grid resolutions are used to represent the same simulation domain. In that case, a better match of the corresponding geometry representations on each level is achieved than with the standard approach, where the domain boundary must align with the cell boundaries. Especially on coarse levels, this cut-cell discretization is essential for obtaining the high rate of convergence of the

multigrid solver, as it makes the discretization equivalent to deriving it from the used restriction and prolongation operators via the Galerkin coarse grid method [TS01]. However, additional expensive sparse matrix-matrix products are not required with this approach.

This makes it easier to update the discretization if, for example, obstacle geometry changes. Note that Eqns. (3.58) and (3.59) can be used both with cut cells or a voxelized domain (with $A_f \in \{0, \Delta x^2\}$) on the fine simulation grid. The restriction operators presented in Section 3.4.3 automatically convert a voxelized domain into a cut cell representation on coarser levels.

3.4.3. Multigrid solver

The non-recursive multigrid V-cycle given in Listing 3.1 consists of five basic operations: restriction ($\mathbf{I}_h^H$), prolongation ($\mathbf{I}_H^h$), SpMV ($\mathbf{L}_l\mathbf{x}_l$), smoothing and the exact solution on the lowest level ($\mathbf{L}_L^{-1}\mathbf{x}_L$). Level 0 is chosen to be the fine grid and L to be the coarsest grid.

```
1  def mgvcycle(x_0, b_0):
2      for l in [0...L):
3          x_l := smooth(L_l, x_l, b_l)
4          r_l := b_l − L_l x_l
5          b_{l+1} := I_l^{l+1} r_l; x_{l+1} := 0
6      x_L := L_L^{-1} b_L
7      for l in (L...0]:
8          x_l := x_l + I_{l+1}^l x_{l+1}
9          x_l := smooth(L_l, x_l, b_l)
10     return x_0
```

Listing 3.1.: Multigrid V-cycle

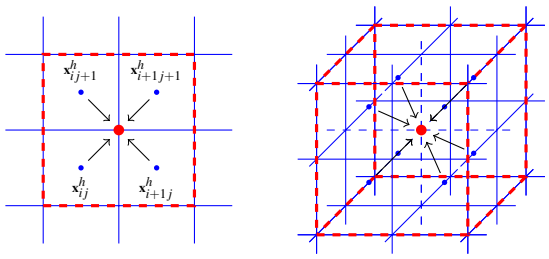
The natural restriction operator $\mathbf{I}_h^H$ resulting from the FV discretization

$$\mathbf{x}_{IJK}^H = \sum_{\substack{i\in[2I,2I+1]\\ j\in[2J,2J+1]\\ k\in[2K,2K+1]}} \mathbf{x}_{ijk}^h \tag{3.60}$$

is a simple sum over four or eight cells in 2D and 3D, respectively, as shown in Fig. 3.23, since the summation over multiple cells is the discrete equivalent to an integral over multiple cells. The prolongation operator $\mathbf{I}_H^h$ is chosen to be the transpose of the restriction operator in matrix representation $\mathbf{I}_H^h = (\mathbf{I}_h^H)^T$, i.e., the coarse grid result is simply repeated for the corresponding fine grid cells.

In fact, these restriction and prolongation operators are not only simpler to implement than the cell-centered trilinear interpolation used by McAdams et al. [MST10], but also result in an improved rate of convergence when used with their algorithm (see Section 3.4.4).

For SpMV, smoothing and exact solution on the lowest level, the matrix $\mathbf{L}$ is not constructed explicitly. Instead the face areas $A_{i\pm\frac{1}{2}jk}$, $A_{ij\pm\frac{1}{2}k}$, $A_{ijk\pm\frac{1}{2}}$ and $A_{d,ijk}^{proj}$ necessary to compute the matrix entries according to Eq. (3.58) are stored similar to the approach described in Section 3.3. For the fine grid, a level set can be used instead of the face areas to reduce the solver's memory footprint. This also helps in unifying the representation of obstacles and free surfaces for the computation of the corresponding areas and extrapolation (see Houston et



(a) (b)

Figure 3.23.: The restriction operator simply sums up the contributions of the fine grid cells, resulting in a cell-centered coarse grid mesh. (a) The two-dimensional restriction for the value $\mathbf{x}^H_{IJ}$ (red dot) on the coarse grid H. The indices for the values on the fine grid h are $i = 2I$ and $j = 2J$. (b) Three-dimensional restriction operator.

al. [HBW03]). However, some details can be lost when recreating face areas from a level set, as compared to directly creating them from the geometry. For coarse grid levels, the corresponding areas can be calculated either from the geometry or from the fine grid areas. For the Dirichlet areas $A^{proj}_{d,ijk}$ the regular restriction operator can be used, as only the total sum per cell is required. The internal face areas can be computed by

$$\begin{aligned} A^H_{I\pm\frac{1}{2}JK} &= \sum_{\substack{j\in[2J,2J+1]\\ k\in[2K,2K+1]}} A^h_{2I+\frac{1}{2}\pm1\,jk} \\ A^H_{IJ\pm\frac{1}{2}K} &= \sum_{\substack{i\in[2I,2I+1]\\ k\in[2K,2K+1]}} A^h_{i2J+\frac{1}{2}\pm1k} \\ A^H_{IJK\pm\frac{1}{2}} &= \sum_{\substack{i\in[2I,2I+1]\\ h\in[2J,2J+1]}} A^h_{ij2K+\frac{1}{2}\pm1} \end{aligned} \tag{3.61}$$

as depicted in Fig. 3.24, i.e., by skipping every second face in the normal direction and summing up groups of four corresponding faces to the fine grid cells. Note that this coarsening scheme is equivalent to directly discretizing the geometry on each level. Furthermore, although small obstacles may be lost as some faces are skipped in Eq. (3.61), domains that include narrow pipes are not an issue, as the face areas would correspond to the pipes' cross section area. If a level set is used for the fine grid it is not possible to simply subsample it to generate the coarse grid discretization, as important features may be lost.

For smoothing, the red-black Gauss-Seidel algorithm shown in Listing 3.2 was chosen, as it is easy to parallelize and it does not require an additional temporary array while achieving similar smoothing results compared to the weighted Jacobi algorithm used by McAdams et al. [MST10].

As the discretization always yields symmetric linear systems, a CG algorithm can be used for the exact solution on the lowest level. Here, the PCG algorithm (see, e.g., [Bri08]) with a Jacobi preconditioner ($\mathbf{M}^{-1} = \mathrm{diag}(\mathbf{L})^{-1}$) is employed. An ICCG approach could be used instead but would force serialization of the code due to the sequential back substitution. The minimum resolution was chosen to be 8 on the lowest level along the smallest axis similar to McAdams et al. [MST10].

All components of the multigrid solver described in this section are trivially parallelizable (only the dot products in the CG solver on the lowest level require a parallel reduction), reducing serial code such as the boundary smoothing used by McAdams et al. [MST10] to a minimum. Both, a CPU-based version using OpenMP as well as a GPU-based version using CUDA were implemented. However, the latter is limited to relatively small

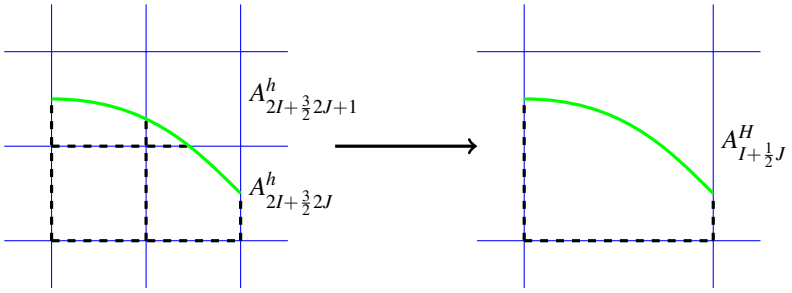


Figure 3.24.: The restriction operator in Eq. (3.61) for obtaining coarse level face areas (lengths in 2D) corresponds to discretizing the geometry on the coarse level.

```
1 def rbsmooth(L, x, b):
2     for c in [0, 1]:
3         for ijk in {ijk ∈ Ω : i + j + k ≡ c mod 2}:
4             x_ijk := (b_ijk − (L − diag(L))_ijk,∗ x) / L_ijk,ijk
5     return x
```

Listing 3.2.: Red-black Gauss-Seidel smoothing

simulation domains of fewer than 12 million cells due to the limited memory available on an individual graphics card (3 GB on current hardware). The CG solver in the GPU version uses the merged kernel approach presented in Section 2.5.3.

3.4.4. Results

In this section, the cut-cell multigrid method is compared with other multigrid schemes and with the commonly used ICCG method. Furthermore, examples are shown that are computed with CPU and GPU implementations of the multigrid solver. Finally, the order of the discretization for scenarios with Neumann and Dirichlet boundary conditions is analyzed.

Comparison with Chentanez and Müller At first, the cut-cell multigrid method is compared with the multigrid solver proposed by Chentanez and Müller [CM11a] with respect to convergence in the L_2 norm. For this comparison, a simulation domain $\Omega = [0\text{m}, 1\text{m}]^3$ with five Neumann boundaries, one Dirichlet boundary and a

Solver	Iterations	Time [s]	Speedup
ICCG	250	160.89	—
Chentanez and Müller	5	17.88	9×
Cut-cell multigrid	2	7.18	22.4×

Table 3.5.: Comparison of duration and iteration count for a normalized target tolerance of $5 \cdot 10^{-3}$ (Multithreading disabled).

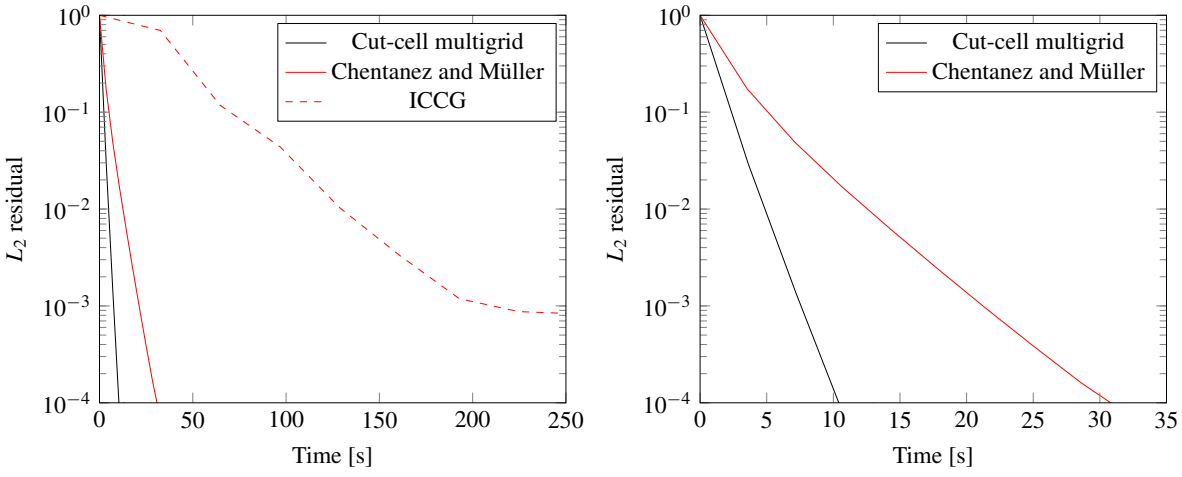


Figure 3.25.: (a) Normalized L_2 residual over time for an ICCG solver, Chentanez' method and the cut-cell multigrid method. (b) The residual of the ICCG solver is omitted to better compare the two multigrid approaches.

Setup	McAdams et al. Unmodified	McAdams et al. Modified	Cut-cell multigrid method
Closed	0.699	0.997	1.419
Free surface	0.735	0.866	1.111

Table 3.6.: Average change in residual magnitude for the setups used in Fig. 3.26 as calculated by $\overline{\log_{10} \frac{\|\mathbf{r}_i\|_2}{\|\mathbf{r}_{i+1}\|_2}}$.

spherical obstacle with a radius of 0.1m at its center was used. The pressure correction for the first simulation step with a uniform initial velocity of $\mathbf{v} = (10,0,0)^T \left[\frac{\mathrm{m}}{\mathrm{s}}\right]$ was computed for a simulation grid of 256^3 cells.

In the following, the rate of convergence for both multigrid approaches and for an ICCG solver are compared. The convergence is measured by determining the residual vector $\mathbf{r}_i = \mathbf{b} - \mathbf{A}\mathbf{x}_i$ in iteration i and evaluating the normalized residual $\frac{\|\mathbf{r}_i\|}{\|\mathbf{r}_0\|}$. This number is used as a threshold to determine whether the solution is sufficiently accurate. For small values, this number is related to the error $\|\mathbf{x}_i - \mathbf{x}\|$. The diagrams in Fig. 3.25 show plots of the normalized residual over time for single-threaded implementations. The data for the ICCG solver was generated by evaluating the residual only every 50 iterations to prevent distorting the performance measurement. The cut-cell multigrid method is faster than the ICCG method by a factor of more than 20 for a normalized residual tolerance of $5 \cdot 10^{-3}$. This factor increases even further when parallelization is enabled with nearly linear scaling as analyzed later in this section. Relative to the multigrid scheme of Chentanez and Müller [CM11b], using the same number of smoothing steps and matrix-vector multiplication, the cut-cell multigrid solver achieves a speedup of approximately 2.5×. For a threshold of 10^{-4} this speedup increases by a factor of three.

Comparison with McAdams et al. To compare the cut-cell multigrid method with the one presented by McAdams et al. [MST10], their scheme was modified w.r.t. position of the Dirichlet conditions. Therefore,

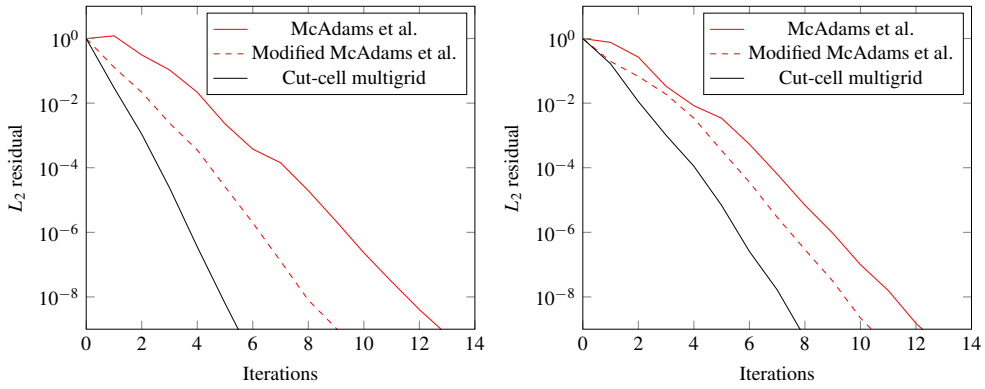


Figure 3.26.: (a) Comparison of the normalized L_2 residual over iterations for a closed domain with two obstacles (b) Normalized L_2 residual for the free-surface example with two obstacles as shown in Fig. 3.27.

Figure 3.27.: (a) Two-dimensional domain for free-surface tests. Dark green areas correspond to Dirichlet boundaries, white areas to Neumann boundaries and blue areas to fluid. (b) Resulting pressure under gravity with colors indicating low pressure in blue to high in red.

the Dirichlet conditions are put directly at the boundaries instead of at the next cell by adding 2 rather than 1 to the main diagonal for neighboring Dirichlet cells. The scheme was reimplemented in Python (2D only). The change in discretization improved the convergence slightly as shown in Table 3.6. In both cases, a convergence rate of slightly more than one order of magnitude per two iterations was achieved as stated in McAdams et al. [MST10]. Both methods used eight passes of damped Jacobi smoothing and were applied as a preconditioner for a CG solver. Additionally, two passes of boundary pre- and post-smoothing were used for McAdams et al.'s approach. As boundary smoothing was implemented as unoptimized, unvectorized Python code, any timings would be severely distorted, therefore the residuals are only given over iterations. Furthermore, McAdams' method was modified to use the same, simpler constant restriction and prolongation operators as described here (see Section 3.4.3) instead of their proposed bilinear operators. This modified version is also included in the comparison.

The residuals of these multigrid schemes are compared in Fig. 3.26. For Fig. 3.26 (b), the setup shown in Fig. 3.27 was used, for Fig. 3.26 (a) an identical setup was used, except that the Dirichlet boundary was limited to the upper border of the 128^2 domain. In both cases, the cut-cell multigrid method achieves a rate of convergence

Scenario	Resolution	lvls	t_{proj}	t_{adv}	t_{all}
Sphere	512×256^2	6	10.5	5.75	19.9
Sphere	256×128^2	5	2.13	0.72	3.33
Car	256×128^2	5	2.68	0.73	3.90
Wasp	256×128^2	5	1.94	0.73	3.16
Dam	256×128^2	5	0.74	0.38	5.29
Droplet	256^3	6	2.76	1.08	18.6
Sphere (gpu)	256×128^2	5	0.93	0.01	0.95

Table 3.7.: This table summarizes the scenarios computed with the cut-cell multigrid method. The resolution, the number of multigrid levels and the computation times in seconds for projection t_{proj}, advection t_{adv} and for one time step t_{all} are given. Note that the free-surface simulations were computed on different hardware. Furthermore, the total step time is significantly influenced by the unoptimized extrapolation and surface reconstruction steps.

of more than one order of magnitude per iteration as shown in Table 3.6. Although the modified solver based on McAdams et al. shows a higher rate of convergence than the unmodified solver, the cut-cell multigrid method maintains a significantly better rate of convergence. This is due to the consistent hierarchy of discretization and a more accurate representation of the solution on different levels of resolution.

Convergence, Performance and Examples Now, several simulations computed with CPU- and GPU implementations of the cut-cell multigrid method are shown. The smoke simulations have been performed on an Intel Core i7 4930 with six cores with an NVIDIA GeForce GTX 970 for the GPU version. All free-surface simulations used a dual processor Intel Xeon E5-2650 system with a total of sixteen cores. The parallel versions were implemented with OpenMP for CPUs and with CUDA for GPUs.

Table 3.7 summarizes the scenarios and gives the respective mesh resolutions. All simulations used eight pre- and post-smoothing steps in the V-cycles. The number of levels is determined such that on the coarsest level, the minimum dimension is equal to or less than eight. The system of linear equations for the pressure correction is solved until convergence. Therefore, the L_2-residual is computed after every V-cycle and a new V-cycle is only executed if the residual improved by more than 30%. Fig. 3.21 shows three frames of a CPU-based smoke simulation on a 512×256^2 grid. The pressure correction for this scenario took 10.5s per frame on average with the proposed cut-cell multigrid method. Fig. 3.29 shows two different free-surface flow problems. For the *Dam break scenario* at 256×128^2 the pressure projection step took an average of 0.74s and for the falling *Droplet scenario* running at 256^3 2.76s were needed. As a direct comparison of the CPU and GPU implementations, a lower-resolution (256×128^2) version of the scenario in Fig. 3.21 was computed both on an i7 4930 and a GeForce GTX 970. The overall computation took an average of 3.3s for the parallel CPU version and 0.95s for the GPU version. By considering the parallel speedup given in Table 3.8, a single-core CPU implementation would require approximately 17s, which corresponds to an acceleration by a factor of 18 for the GPU version. The computation times for all scenarios are given in Table 3.7.

The two-dimensional simulations in Section 3.3 have shown that the solution of the linear system with the CG solver is clearly the bottleneck. The tests were repeated comparing the computation time between the cut-cell multigrid method and the CG solver. The table in Fig. 3.28 lists the time required to reduce the residual of the linear system by a factor of 10^{-5} with varying resolution $n \times n$. The graphs in Fig. 3.28 show the significantly reduced computation times of the multigrid method and the linear scaling of computation time w.r.t. number of degrees of freedom.

n	# cells	t_{cg}	t_{mg}
32	1024	6.85	7.24
64	4096	13.02	9.09
128	16384	31.01	11.42
256	65536	84.48	25.25
512	262144	491.70	48.84
1024	1048576	3526.54	122.79

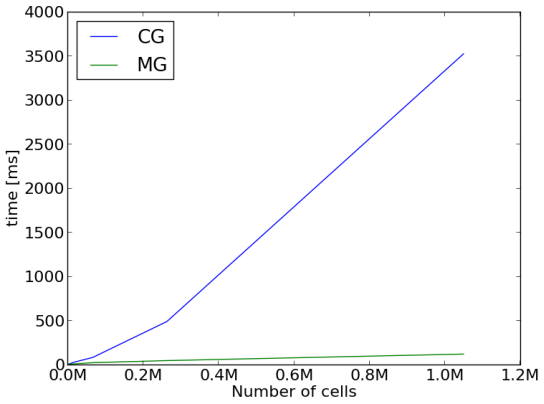


Figure 3.28.: The analysis of the two-dimensional scenario in Section 3.3.5.1 was repeated with the cut-cell multigrid method. The multigrid solver shows a clear performance benefit and computing times scale linearly with the number of degrees of freedom.

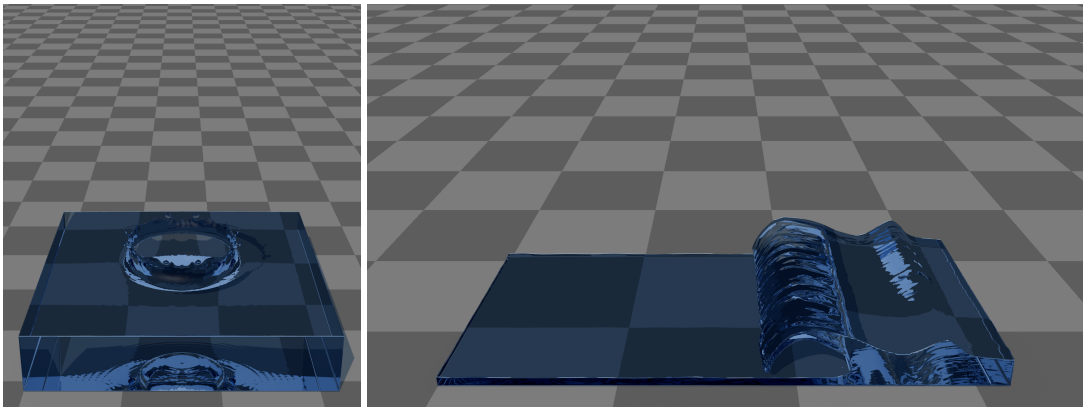


Figure 3.29.: Screenshots from the two free-surface simulations that were run at $256 \times 256 \times 256$ and $256 \times 128 \times 128$, respectively.

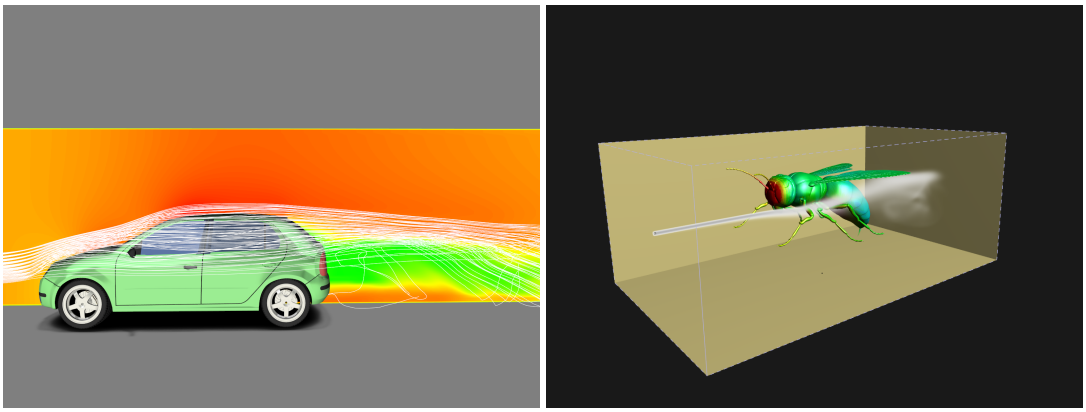


Figure 3.30.: Two screenshots of the GPU implementation which integrates interactive simulation and real-time scientific visualization showing a car and a wasp in a virtual wind tunnel.

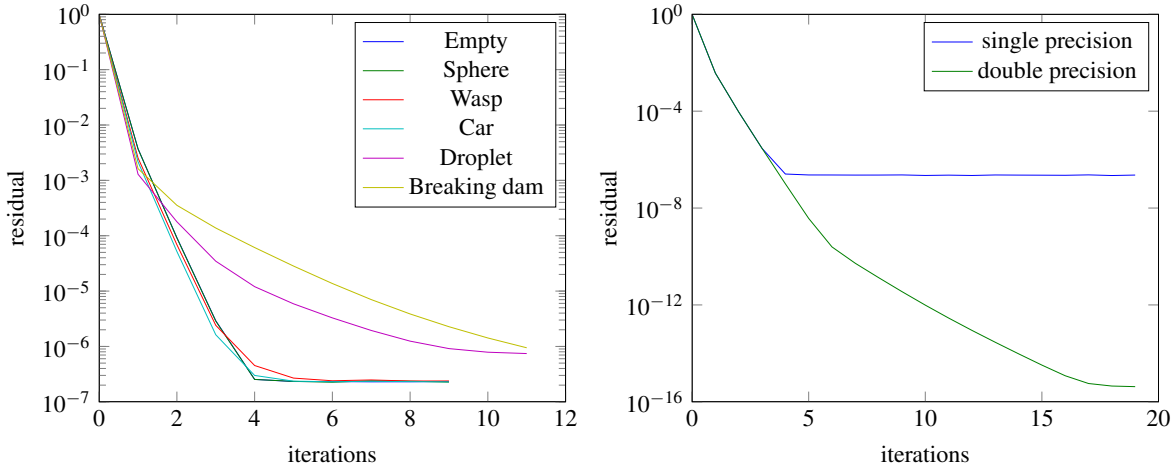
# threads	1	2	3	4	5	6
total time (ms)	24.0	12.3	8.3	6.3	5.2	4.6
speedup	1.0	1.9	2.9	3.8	4.6	5.2

Table 3.8.: Average computation times for one time step in a scenario similar to Fig. 3.21 with a resolution of 256×128^2 using a varying number of threads. The measurements in Table 3.7 used hyper-threading with 12 threads.

Fig. 3.30 shows two screenshots from an interactive wind tunnel environment with a 200×64^2 grid which integrates simulation and real-time scientific visualization. In order to achieve a high frame rate for this interactive scenario, the result from the last pressure correction was reused as a first guess and the stopping criterion was relaxed to a relative improvement of $2 \cdot 10^{-4}$ for the residual in the L_2 norm. In this case, the pressure correction took approximately 28ms per frame on an NVIDIA GeForce GTX 580. When further increasing the grid resolution to 256×128^2 for the scenarios in Fig. 3.30, the computation of a single simulation step takes less than 100 ms and therefore interactive rates for more than four million cells are achieved.

Table 3.8 shows the parallel speedup for a scenario similar to Fig. 3.21 with a resolution of 256×128^2. On an Intel Core i7 4930 with six cores the runtime was evaluated by varying the number of threads. The resulting speedup is almost linear.

Fig. 3.31 (a) shows convergence plots for representative pressure correction steps of the scenarios listed in Table 3.7, where the stopping criterion was removed. The diagrams demonstrate stable and comparable convergence behavior independent of mesh resolution and scene complexity for all scenarios. The difference in convergence between simulations with and without a free surface can be explained by the different discretization order analyzed in the next paragraph. As Dirichlet cut-cells result in first order accuracy, the geometry might be less well resolved on coarser levels of the hierarchy leading to weaker convergence. However, the cut-cell multigrid approach is especially well suited for flows without free surfaces but it can handle these scenarios as well. Note that single precision is used in all the computations and the residuals do not improve by more than seven orders of magnitude. In order to demonstrate that this effect is due to limited precision, additionally a convergence test was performed using double precision for the *Sphere scenario*. Fig. 3.31 (b) provides a comparison and shows convergence comparable to the state of the art for the more accurate case.



(a) (b)

Figure 3.31.: (a) Convergence plots for the scenarios listed in Table 3.7 using the multigrid algorithm without stopping criterion. (b) Comparison of convergence between single and double precision.

	Error			Order		
Size	$\mathbf{L}_1$	$\mathbf{L}_2$	$\mathbf{L}_\infty$	$\mathbf{L}_1$	$\mathbf{L}_2$	$\mathbf{L}_\infty$
16^2	8.38×10^{-4}	1.35×10^{-3}	5.15×10^{-3}	—	—	—
32^2	2.08×10^{-4}	3.30×10^{-4}	1.50×10^{-3}	2.01	2.03	1.78
64^2	5.13×10^{-5}	8.10×10^{-5}	3.95×10^{-4}	2.02	2.03	1.93
128^2	1.27×10^{-5}	1.99×10^{-5}	9.81×10^{-5}	2.02	2.02	2.01
256^2	3.15×10^{-6}	4.94×10^{-6}	2.41×10^{-5}	2.01	2.01	2.02

Table 3.9.: Discretization order for Neumann boundaries. When halving the cell sizes, the error is reduced by a factor of four showing second order accuracy of velocity in all norms.

Discretization Order In the following, the order of accuracy of discretizations with Neumann and Dirichlet boundaries is analyzed. A scheme is of p-th order accuracy, if the error is proportional to the p-th power of the cell spacing. The order of accuracy of the cut-cell discretization for Neumann boundaries was verified using the problem proposed by Ng et al. [NMG09]. There, a *Hodge decomposition* on a two-dimensional elliptic domain was computed and compared to an analytic solution. In all three norms, L_1, L_2 and L_∞, an order of two for the accuracy of the pressure gradient is achieved as shown in Table 3.9. For Dirichlet boundaries a similar scenario proposed in the work of Li [Li98] was used. However, the numbers in Table 3.10 show that for this kind of boundary conditions only an order of one is achieved. This also is in line with the slightly reduced convergence rates of the residuals depicted in Fig. 3.31 (a), where the boundary conditions cannot be resolved as accurately as in the Neumann scenario. However, the visual effect of an inexact solution at Dirichlet boundaries is far smaller than for Neumann boundaries for smoke simulations. Therefore, in this case, the efficiency of the solver takes precedence.

	Error			Order		
Size	$\mathbf{L}_1$	$\mathbf{L}_2$	$\mathbf{L}_\infty$	$\mathbf{L}_1$	$\mathbf{L}_2$	$\mathbf{L}_\infty$
16^2	1.02×10^{-2}	1.14×10^{-2}	1.67×10^{-2}	—	—	—
32^2	5.11×10^{-3}	5.72×10^{-3}	9.24×10^{-3}	0.995	0.997	0.856
64^2	3.10×10^{-3}	3.49×10^{-3}	6.02×10^{-3}	0.721	0.715	0.617
128^2	1.60×10^{-3}	1.81×10^{-3}	3.50×10^{-3}	0.949	0.946	0.786
256^2	8.38×10^{-4}	9.46×10^{-4}	2.05×10^{-3}	0.937	0.935	0.771

Table 3.10.: Discretization order for Dirichlet boundaries. When halving the cell sizes, the error is reduced by a factor of two resulting in first order accuracy.

3.4.5. Discussion

McAdams et al. [MST10], who were among the first to use a multigrid solver for fluid animation, only achieved convergence with either a large number of smoothing steps and additional boundary smoothing or by embedding the multigrid solver as a preconditioner within a CG solver. Chentanez and Müller [CM11a] improved this by using a variational discretization with compatible discretizations on all levels. None of these methods achieve second order accuracy in all norms for Neumann boundaries. This was addressed by Ng et al. [NMG09], who handle Neumann conditions accurately in an FV setting. Here, this approach was extended by showing how to incorporate less accurate Dirichlet boundaries while retaining symmetry in the linear systems at the same time. Furthermore, the cut-cell approach was embedded in a multigrid solver. In contrast to other methods, such as Ferstl et al. [FWD14] who use a finite element discretization and locally generate a tetrahedral mesh for partially filled cells, cell duplication is avoided to maintain cache coherency and memory alignment.

The performance of the solver was tested with a varying number of smoothing steps. For the smoke simulations the best performance was achieved choosing eight smoothing steps. However, the multigrid scheme also achieved convergence with as little as two steps.

When used on the fine simulation grid, the geometric cut-cell method has difficulties representing thin structures such as cloth, etc., as an infinitesimally thin structure will not reduce the fluid area available for exchange between cells. In the experiments, the multigrid convergence did not suffer from this restriction, e.g., the convergence of the *Wasp scenario*. Variational formulations suffer from similar limitations. In both cases, a conservative voxelization can be used as a fallback solution for thin structures. Compared to purely voxelization-based methods such as McAdams et al. [MST10], the cut-cell multigrid method requires an increased memory bandwidth due to three fluid face arrays being read as opposed to a single integer domain type array.

3.4.6. Summary

A novel multigrid scheme based on a cut-cell FV discretization for fluid simulations has been presented. It has an improved convergence rate of more than one order of magnitude per iteration and a speedup of up to a factor of three for Neumann boundaries compared to state-of-the-art multigrid algorithms. This is due to the discretization which is compatible on all levels. Furthermore, a method to handle Dirichlet boundaries for free-surface simulations in an FV setting was introduced that results in symmetric matrices albeit with slower convergence. The cut-cell multigrid method is simple to implement and fully parallelizable. It does not require special boundary treatment or the embedding as a preconditioner to maintain convergence. The representation of the linear systems is realized without explicitly storing the sparse matrices.

Method	Publication and section	Main result
Geometric cut-cells	[WPnSSF11], Section 3.3	More accurate representation of embedded solid geometry enabling geometry-aligned fluid flow
Implicit matrix data structure	[WPnSSF11], Section 3.3	Acceleration of GPU-based SpMV by 20%
Cut-cell geometric multigrid method	[WMRA*15], Section 3.4	Improved convergence rate of a multigrid solver by a factor of three in comparison to existing multigrid approaches.

Table 3.11.: This table shows the contributions presented in this chapter and summarizes the corresponding results. Furthermore, it provides a reference to the corresponding publication and section, where each method is described.

3.5. Contributions and conclusions

In this section, the contributions made to fluid simulation are summarized. Additionally, the findings and limitations of the novel approaches are discussed.

3.5.1. Contributions

In this section, the following novel methods and data structures for efficient simulation of fluid dynamics have been presented that provide significant speedup and an increase in accuracy in comparison to previous methods.

1. The *geometric cut-cell method* more accurately represents solid geometry embedded on regular grids.
2. The *implicit matrix data structure* for regular grids reduces the amount of required data transfers to the GPU kernels.
3. The *cut-cell geometric multigrid method* significantly improves the convergence rate for the solution of the linear system.

Table 3.11 summarizes the contributions of this chapter, provides references to the publications and lists the individual improvements of each developed method.

The geometric cut-cell method is able to model sub-grid accuracy for embedded geometry. This is achieved by explicitly computing the intersections between the geometry and the regular grid discretization. The face fractions are incorporated into the equations and result in an improved capturing of geometric details. Flow along non-aligned geometry can be simulated more realistically.

Exploiting massively parallel computations enables to significantly speedup the computations for fluid simulation. When considering the special access patterns to GPU memory, the use of bandwidth can be optimized to achieve high efficiency. In order to reduce the number of required load operations, the implicit matrix data structure has been developed that provides a way of encoding the linear system by face fractions. Using this representation accelerates SpMV by 20% and therefore allows to solve the linear systems more quickly. In comparison to a CPU-based approach, the algorithm is 14 times faster on GPUs.

The *cut-cell geometric multigrid method* is building upon the geometric cut-cell representation in order to provide a consistent discretization w.r.t. different resolutions. This results in an efficient algorithm to solve the system of linear equations for computing the pressure term. Thereby, the compute intensive pressure correction

can be solved in less time, significantly speeding up the overall simulation. The resulting improvement in convergence rate and acceleration in computation time is in the order of a factor of three in comparison to other multigrid approaches. In comparison to an ICCG algorithm the solution of the linear system is accelerated by a factor of 22. As all building blocks can be executed in parallel, an efficient GPU version is possible. Along with a relaxed convergence criterion, interactive simulation with more than four million grid cells can be achieved.

3.5.2. Discussion

The novel methods and data structures presented in this section improve the state of the art for fluid simulation by yielding significant acceleration and more accurate flow. The goal to significantly increase the accuracy and the number of degrees of freedom for interactive fluid simulation has been achieved. With a constant computing time budget, the resulting speedups allow for more detailed modeling of the fluid flow due to the opportunity to use higher spatial resolution. At the same time, the sub-grid modeling on computational attractive regular grids provides more detailed flow at fluid-solid boundaries as embedded geometry can be resolved with higher accuracy. Interactive rates are achieved with a GPU-based CG algorithm even when accurately solving the linear systems of the pressure correction for moderate grid resolution. By employing the GPU-based, highly convergent, cut-cell geometric multigrid method, interactive rates are possible even with a comparatively high resolution of up to four million elements.

The *geometric cut-cell method* enables modeling of sub-grid accuracy on regular grids. Thin domains such as narrow pipes can be modeled and simulated, as the formulation is based on fluxes. However, when incorporating thin solids that split cells into two parts the cut-cell approach in its current version is not able to prevent flow through the geometry. This is due to the fact that only one pressure sample is used for both sides of the solid geometry due to the grid topology. Although not accurate, a solution to this problem with the same resolution is to thicken the geometry to ensure that one cell is used for two different domains. Duplicating the cells in these cases would allow the correct simulation of this behavior, but the computationally attractive regular grid topology would be lost.

The option of using massively parallel computing on powerful GPU hardware yields significant performance improvements. However, to achieve interactive simulation several aspects must be considered. First, highly efficient numerical methods are required to solve the governing equations in a short amount of time. For the computationally intensive solution of linear systems, limits on the iterations or comparable moderate residual thresholds are options to obtain low computing times. Additionally, specialized GPU data structures that respect the required memory access patterns for efficiently exploiting the high computational power are necessary to further speed up the computations. A further key ingredient to obtain fast speedups in interactive simulation is to keep the computed data in GPU memory. This avoids the significant performance penalty when transferring data back and forth between GPU and CPU memory spaces. This is especially beneficial if the objective is to achieve visual interactive simulation applications, where the computed data on the GPU can be directly used for visualization.

Multigrid methods are an attractive option for solving large linear systems when computing the pressure correction on highly detailed grids as they yield linear time complexity w.r.t. the number of degrees of freedom. However, for smaller grid resolutions CG method are a better alternative as they converge more quickly with a small number of unknowns w.r.t. computing time. A notable fact for both types of solvers is that in time dependent simulations the errors that are induced when not solving the linear systems accurately are compensated in subsequent time steps. Reusing the previously computed solution for the pressure correction provides a good initial guess when assuming that the fields do not change significantly.

For FD and FV discretizations, previous works developed several methods to model sub-grid accuracy for solid and free-surface geometry. The GFM on FD discretizations provides second order accuracy for Dirichlet boundary conditions to more accurately capture the free surfaces on a sub-grid level. However, sub-grid accuracy with Neumann boundary conditions and symmetric positive definite matrices can only be achieved by introducing complex stencils that increase the memory consumption and complicate the construction of efficient multigrid methods. In contrast, the developed cut-cell FV method has complementary properties, as it shows second order accuracy for Neumann boundaries and sub-grid accurate Dirichlet boundaries are difficult to achieve. Early approaches for Dirichlet boundaries resulted in non-symmetric matrices that are numerically more challenging to solve. The FV Dirichlet condition developed in the context of the *cut-cell geometric multigrid method* maintains the symmetry in the linear systems. However, the approach is only first order accurate. Therefore, the cut-cell method is better suited for scenarios with pure Neumann boundaries, i.e., simulation scenarios without free surfaces.

Computing on GPUs, especially on commodity hardware, requires the use of single precision to represent floating point numbers, as the computational power significantly decreases when switching to the more commonly used double precision. The significant match of the residuals for linear solvers between using single and double precision is remarkable. However, the maximum achievable residual improvement varies clearly. With single precision sometimes only three to four orders of magnitude of residual reduction can be achieved, whereas double precision can reach more than ten orders of magnitude. This has to be taken into account when specifying limits for the residuals, especially when using single precision that is required on commodity GPU hardware. For future work, it is therefore worth investigating which components require double precision and where single precision is sufficient in order to achieve high convergence rates and minimize the data size at the same time.

As mentioned above the development of specialized data structures is always required to exploit the power of GPUs, as the use of memory bandwidth has to be maximized to obtain significant accelerations. This is an essential difference in comparison to CPU data structures, where a lot of cache hierarchies are able to automatically hide the latency of the memory accesses. However, with newer GPU hardware caches are growing. For early GPU hardware it was very important to exactly conform to the proposed access patterns, whereas newer chips such as NVIDIA's Fermi, Kepler or Maxwell are not that susceptible to non-optimal access. Taking care of memory access patterns for GPUs is still important but the rules to achieve efficiency are relaxed. However, with newer hardware it is possible to write less optimized code that also achieves high speedups as the architectures are designed to alleviate non-regular access patterns.

4. Conclusions and future work

The objective of this thesis to develop faster, more accurate and more efficient algorithms for physically based simulation has been achieved with several contributions. The novel methods have been applied in two distinct physical domains both highly relevant to computer animation. They improve the state of the art by significantly accelerating computations and delivering more accuracy. The central contributions of this thesis have been summarized in Section 2.6 and 3.5 but are briefly listed here again and depicted in Fig. 4.1. The performance benefits and the corresponding publications are listed in the Tables 2.12 and 3.11. In the following, the research questions are reviewed and related to the novel developments. The approaches of both physical domains are discussed and conclusions are drawn. Furthermore, new possibilities for future work developed during research. This is summarized in Section 4.3.

4.1. Research questions

The goal of this thesis to contribute to faster and more accurate simulations has been addressed by novel methods associated with the three pillars of this thesis: *more accurate discrete representations*, *efficient methods for linear systems*, and *data structures and methods for massively parallel computing*. The research questions associated with the three pillars have been addressed in the following way:

In the category *more accurate discrete representations*, the *quadratic and cubic Bernstein-Bézier-form (BB-Form) methods* have been developed to enable more detailed deformation simulations due to a higher-order approximation of the simulated displacement fields. It has been shown that this type of representation can reproduce motion with more detail. Furthermore, higher-order representations are able to improve the accuracy of near-incompressible solids. At the same time, this accelerates the overall simulation process as fewer elements are required to reproduce the desired behavior answering the research question:

> 1.1 Are higher-order BB-form polynomials beneficial for improving quality and speed of physically based simulation?

Furthermore, the *geometric cut-cell method* for finite volume (FV) discretizations has been introduced to provide sub-grid accurate modeling of embedded geometry on computationally beneficial regular grids. Especially, flow along geometry that is not aligned with the discretization can be reproduced more accurately giving a positive answer to the research question:

> 1.2 Can cut cells improve the quality of fluid simulations on regular grids?

Additionally, the cut-cell discretization enabled the development of the efficient *cut-cell geometric multigrid method* to solve the linear systems in the pressure correction. Furthermore, Dirichlet boundary conditions in an FV setting for free-surface simulations that maintain symmetry in the linear systems and result in compatible multigrid hierarchies have been developed.

Multigrid and other methods have been proposed in the category *efficient methods for linear systems*. The *p-multigrid method* was developed to accelerate the solution process of linear systems generated by higher-order finite element (FE) discretizations. Based on a polynomial hierarchy, the linear system is successively solved on lower-order discretizations leading to an improved rate of convergence. It has been demonstrated that linear

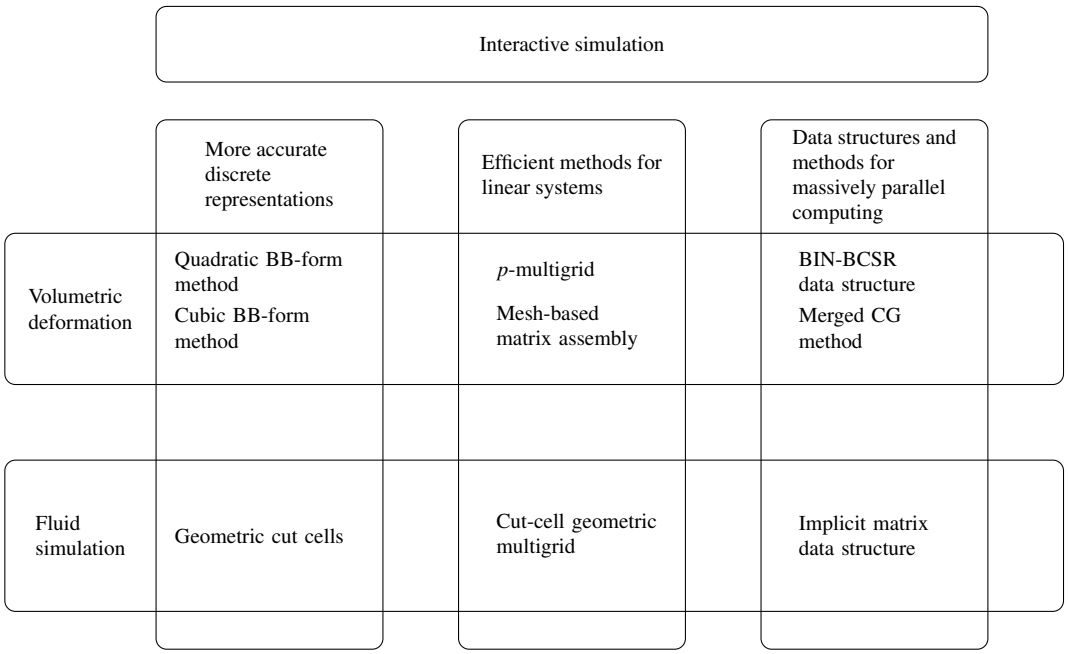


Figure 4.1.: The contributions to address the challenge of interactive simulation are associated with three pillars: *more accurate discrete representations*, *efficient methods for linear system*, and *data structures and methods for massively parallel computing*.

systems can be solved much more quickly than with the commonly used preconditioned conjugate gradient (PCG) algorithms giving evidence to the question:

2.1 Can efficient multigrid algorithms be constructed based on polynomial hierarchies?

The *cut-cell geometric multigrid method* builds upon the cut-cell discretization and solves the linear system for the pressure correction in fluid simulations with improved convergence rates. The building blocks are fully parallelizable, scale well on graphics processing units (GPUs) and multi-core CPU architectures and therefore result in a significant speedup. High convergence rates and therefore accelerated computation times answer the question:

2.2 How do cut-cell discretization hierarchies improve the convergence rates of multigrid algorithms ?

Additionally, the *mesh-based matrix assembly* has been developed to accelerate the reassembly of the linear system, which is required in corotational deformation simulations in every time step. This gives a positive answer to the research question:

2.3 How can matrix assembly be accelerated?

The pillar *data structures and methods for massively parallel computing* comprises methods that have been developed to efficiently make use of the computational power offered by GPUs. They provide a significant speedup in comparison to CPU implementations. In addition, efficient GPU data structures have been designed to improve the memory access patterns and to better exploit the memory bandwidth in comparison to existing GPU approaches. The *BIN-BCSR data structure* has been introduced to optimize the access patterns for matrix

assembly and sparse matrix-vector multiplications (SpMV). This accelerates the construction and solution of the linear systems generated from unstructured FE meshes for deformation simulation. Furthermore, the *implicit matrix data structure* has been designed for cut-cell geometry representations on regular grids for fluid simulations. This data structure reduces the amount of data that must be loaded by the GPU kernels and therefore allows for faster SpMV operations and faster solution of linear systems. These data structures are an immediate answer to:

3.1 What are efficient data structures for simulation on GPUs?

Finally, a GPU-based deformation simulation has been developed, where all computations such as polar decompositions, matrix assembly and solution of linear systems are performed on the GPU. In addition, a GPU-based fluid simulation method was developed. It has been demonstrated that both types of simulation can be significantly accelerated in comparison to CPU implementations. Massively parallel versions of all building blocks have been realized, answering the question:

3.2 How can existing simulation methods be adapted to exploit the computational power of GPUs?

4.2. Conclusions

In this thesis, various approaches for simulation of dynamic volumetric deformation and fluid simulation have been developed and investigated. The novel methods presented in this thesis provide a significant acceleration for physically based simulation. The goal of increasing accuracy in simulations while reducing computation time has been achieved. The limit of mesh resolution for interactivity has been extended. The conclusions specific to each physical domain have already been discussed in Section 2.6 and 3.5. In the following, the implications of the novel methods are reflected especially with their classification into the respective pillars. Furthermore, the differences between the physical domains in terms of discretization, performance and suitability for massively parallel computing are discussed.

The first pillar of this thesis is *more accurate discrete representations* as depicted in Fig. 4.1 on the left. The methods developed in this context can represent the same richness of detail w.r.t. the physical quantities with fewer elements. In the domain of volumetric deformation simulation, higher-order polynomials in BB-form have been applied to model the displacement field. More specifically, quadratic and cubic finite element bases that require far fewer elements to reproduce the desired deformation have been constructed and applied to simulation. Although the number of degrees of freedom per element is significantly higher, the overall number of degrees of freedom and therefore the dimension of the resulting linear system are lower. In one of the tests with quadratic bases, an acceleration factor of roughly five was achieved mainly due to the faster solution of the smaller linear system. Similar conclusions can be drawn when applying *geometric cut cells* on regular grids for fluid simulation. In this case, geometric boundaries can be represented with far fewer cells compared to a binary discretization also resulting in smaller linear systems. Artifacts on regular grids are significantly reduced by the adoption of the *geometric cut-cell discretization*, as the geometry can be resolved more accurately. In summary, the novel methods associated with the first pillar produce smaller linear systems in comparison with the commonly used methods and therefore accelerate the overall solution process.

The second pillar *efficient methods for linear systems* encompasses methods that focus on the setup and the solution of linear systems. The *mesh-based matrix assembly* for higher-order finite element methods was developed to accelerate the construction of linear systems. In volumetric deformation simulation based on corotational elasticity, the matrices must be reconstructed in every time step . The novel method reduces the assembly time by 75%. In addition, multigrid methods for solving the corresponding linear systems have been the focus of the second pillar. Multigrid methods are known to be the fastest class of linear solvers, as they exhibit linear complexity w.r.t. the number of degrees of freedom in contrast to other approaches. However, achieving high

convergence rates for multigrid methods is heavily dependent on the consistency of the discretizations on different levels of resolution, i.e., the similarity of geometric representation. The novel *p-multigrid method* builds upon hierarchies of polynomial representations to efficiently solve linear systems in the domain of volumetric deformation simulation with higher-order elements. The discretizations on lower levels correspond to a formulation with lower polynomial degree. These representations are defined w.r.t. the same tetrahedral mesh, which implies a high consistency between the levels and therefore high convergence rates. Tests with higher-order finite elements for volumetric deformation simulations have shown accelerations by a factor of roughly seven compared to a PCG algorithm. In the domain of fluid simulation, a specialized multigrid solver was constructed that takes the compatibility of discretizations on different levels of resolution into account. Hence, the *cut-cell geometric multigrid* method was developed to efficiently solve systems of linear equations for the pressure correction on regular grids. Based on geometric cut cells, a representation of the geometry on different levels of resolution is devised, which is more consistent in comparison to a binary discretization. In comparison to standard methods such as the incomplete Cholesky preconditioned conjugate gradient method (ICCG), the algorithm is faster by a factor of 22 in one of the tests running on a single core. In comparison to other multigrid methods that build on different types of discretizations, an acceleration by a factor of roughly three was achieved. In summary, the novel methods associated with the second pillar accelerate the solution process and the setup of the linear systems.

The third pillar *data structures and methods for massively parallel computing* is concerned with methods that make use of parallelization, especially exploiting massively parallel processors. Parallelization in general is a viable option in the era of stagnating processing speed on single cores and the emergence of multiple cores on a single processor chip. Especially on GPUs, the characteristics of the parallel hardware, the execution model and memory access patterns have to be taken into account in order to efficiently utilize this increase in number of cores and computing power. This can be ensured by designing specialized algorithms and data structures that respect these requirements. In the domain of volumetric deformation simulation, the *BIN-BCSR data structure* has been developed to enable fast SpMV operations that are required for an efficient solution of the corresponding linear systems. The proposed data structure exhibits an optimized memory layout to allow for optimal parallel access by thread groups. Therefore, the data is reordered accordingly and the block-structure inherent to deformation simulation is exploited to save memory bandwidth. Thereby, an acceleration for SpMV operations by a factor of two to three in comparison to standard GPU data structures is achieved. In order to reduce the time required to solve linear systems on the GPU, the *merged CG method* has been developed. This method reduces the number of kernel calls by combining several operations. Synchronization between kernel executions is realized with a two-step reduction approach, where in the first step the data is reduced locally and the final computation is conducted in a second kernel. As each kernel execution comes along with a constant overhead, the avoidance of unnecessary calls especially pays off for moderately sized discretizations that are well suited for interactive applications. Depending on the size of the linear system, an acceleration of three to four is achieved. For discretizations on regular grids, the *implicit matrix data structure* was designed to reduce the amount of memory required for the matrix in comparison to a standard five or seven point Laplace stencil storage. Employing this data structure for SpMV operations or Jacobi iteration results in an acceleration of 20% on average. In summary, the data structures and methods associated with the third pillar focus on the efficient exploitation of massively parallel architectures to further accelerate computations.

The three pillars in this thesis represent different avenues to increase the ratio between accuracy and computation time. The methods in the respective pillars consider distinct aspects but share the focus on the solution process of linear systems. As described above, each of the pillars comprises methods that have different types of implications on the solution process. The first pillar represents more accurate modeling, which generally results in smaller linear systems that can be solved faster. In the second pillar, the corresponding methods accelerate the setup or the solution process itself. Data structures and methods associated with the third pillar focus

on efficiently exploiting parallel architectures and leveraging the additional computing power offered by GPUs. Conceptually, the three pillars are independent of each other, implying that the corresponding methods can be combined if compatible. One example for such a combination is the massively parallel fluid simulation with the *cut-cell geometric multigrid* method and the *implicit matrix data structure* that make use GPU acceleration, high convergence rates and accelerated matrix operations. Another example is the GPU-accelerated volumetric deformation simulation that combines the *quadratic BB-form method*, the *BIN-BCSR data structure* and the *merged conjugate gradient method*. In both examples, there are synergies of the individual method's benefits that result in an even higher increase of the ratio between accuracy and computation time.

The two application areas in this thesis, the physical domains volumetric deformation simulation and fluid simulations, exhibit fundamental differences in terms of discretization, performance and suitability for massively parallel computing. Although the derivation is based on the same principles and the resulting PDEs are similar, there are several differences such as the applied algorithms and types of discretization. Usually, deformation simulation algorithms are formulated in a Lagrangian perspective, i.e., the mesh moves during the simulation, while in fluid simulations an Eulerian viewpoint with stationary meshes is taken. Furthermore, in deformation simulation scenarios, the geometric shape is of importance, i.e., how does the boundary of the object change over time. In contrast, for fluid simulations the visually interesting effects arise and develop within the simulation domain or at free surfaces. This is also reflected by the fact that different grid types are used. Regular grid discretizations have a constant cell density. This is beneficial for fluid simulations as it is not known a priori where visually interesting motion will occur. Tetrahedral discretizations are able to adaptively conform to the boundary of the geometry making it attractive for deformation simulations. These types of meshes can be more generally classified into structured and unstructured grids. Structured grids have a regular topology, i.e., adjacency relations are implicitly given. In contrast, unstructured grids have irregular neighborhood relationships and the connectivity information must explicitly be stored.

Thus, there are significant differences in terms of performance especially on GPUs between these discretizations. More specifically, the number of elements that can be simulated in interactive time with at least ten updates a second vary heavily between the two physical domains. Although both simulation applications in this thesis are not directly comparable as the algorithms differ, conclusions can be drawn from the performance of the solution of linear systems with the different grid types. In Section 3.4, the GPU-based fluid simulation achieved interactivity with up to four million elements on a regular grid discretization. In contrast, the FE discretizations on tetrahedral meshes achieved around 100k linear and 13K quadratic FEs in interactive time, as reported in Section 2.5. This is a difference in number of elements of more than one order of magnitude. Of course, both approaches differ significantly in the algorithmic building blocks, however, the solution of a linear system is the bottleneck in both cases. The reasons for the performance differences are manifold. First, for single-phase fluid simulations with static geometry, the linear system is constant throughout the whole simulation. Therefore, no matrix reconstruction is required and multigrid hierarchies do not have to be updated. In contrast, corotational elasticity requires performing per-element polar decompositions and updating the stiffness matrix in every time step. The algorithms in this thesis for solving linear systems require SpMV or similar operations. For structured grid discretizations, these operations do not require column information as it is implicitly given by the regular mesh topology. The number of non-zero entries per row is constant with seven or five entries for the discretization schemes used in this thesis. Furthermore, loading entries from the input vector benefits from cache coherent memory access patterns. In contrast, these access patterns are irregular for unstructured grids. In addition, matrices that arise from unstructured grids require to load additional data that represents the matrix structure, i.e., the column information for the non-zero entries. Moreover, a FE discretization of elasticity induces a 3×3 block for each pair of nodes. The number of nodes that share at least one tetrahedron determine the number of non-zero entries in the corresponding row. Even for linear FEs, this number is comparatively higher and SpMV operations are computationally more expensive. Therefore, a significantly larger number of elements can be simulated in

interactive time with structured grids on GPUs as this type of data structure is better suited for massively parallel computation. Furthermore, the numbers stated above were generated using a GPU-accelerated multigrid and CG approach for fluid and deformation simulation, respectively. Although the possibility of a massively parallel version of the *p-multigrid algorithm* has not yet been investigated, there are differences in the convergence behavior of the *cut-cell geometric multigrid* and the *p-multigrid approach*. In general, multigrid algorithms exhibit higher convergence rates on regular grids than on irregular discretizations. This is due to the fact that smoothing operations depend on element sizes and work best on uniform discretizations. In the case of the *p-multigrid algorithm*, the depth of the hierarchy is implicitly bounded by the polynomial degree. Therefore, a comparatively large linear system has to be solved on the lowest level, implying a higher computational effort.

In order to accelerate computations, parallelizations on both multi-core CPU and GPU architectures have been realized. The experiments have shown that there is a significant speedup by a factor in the order of 20 when execution takes place on GPUs in comparison to single-core CPU implementations. Utilizing parallel computations on multi-core CPUs also results in reasonable acceleration, as the speedup scales almost linear with the number of processor cores for many algorithms. Therefore, in comparison to multi-core CPU implementations the GPU acceleration is only moderate with factors between three to six depending on the problem to be simulated and the hardware. However, for interactive applications it is important that the computed results in form of geometry or simulation data is directly visualized. Therefore, it is beneficial that GPU computations do not require additional data transfers that would induce a further performance penalty.

4.3. Future work

There are many different aspects that provide interesting avenues for future research. These can be classified into higher-order approaches, improvements for corotational elasticity, research regarding accuracy, further GPU approaches and coupling of both domains.

4.3.1. Higher-order approaches

Cuts and fracture for deformable objects are extensions that have been already developed for linear FEs. One question is how cuts and fracture can be incorporated with higher-order FEs. In contrast to linear elements, higher-order bases allow for more detailed geometry and offer the possibility to better represent curved cuts. Furthermore, other types of material such as anisotropic or non-linear material could also benefit from modeling with quadratic or cubic BB-form elements. Although these methods have already been developed for standard linear FE, it is worth investigating if higher-order representations can also improve the simulation quality with different material models. Another question is how to handle collision response of higher-order FEs. As the deformed geometry can be curved, a more accurate collision response taking into account non-flat surfaces is likely to improve the realism of motion when colliding with other geometry.

An interesting path for future work is the investigation whether higher-order FEs can also improve the modeling of fluid dynamics. The degrees of freedom in the FE mesh could be advected using a method comparable to semi-Lagrangian advection. Then, it is possible to solve the pressure correction with fewer elements due to the higher approximation order. Especially for free surface simulations, the higher-order approach can be an interesting option for representing curved liquid surfaces, which are implicitly given when using a higher-order polynomial representation.

4.3.2. Corotational elasticity

The main approach applied in this thesis for volumetric deformation is the corotational method, which does not take the dependency of the rotation matrix on the vertex position into account. The so-called exact corotational method includes the dependency in its formulation and leads to more accurate simulations. However, it has so far only been developed for linear FEs. The combination of higher-order FEs with this extended modeling would be interesting for further investigation.

The original corotational method might be improved in terms of performance based on the following observations. In comparison to linear geometric modeling, the major computational effort is caused by the corotational formulation. The polar decomposition, the update and assembly of the linear systems consume the major part of time in the simulation algorithms besides the solution of the linear systems. Therefore, a novel formulation for physically based deformation simulation with constant stiffness matrices, which simultaneously maintains the volume conserving properties, would be a significant improvement. An interesting question in this context is whether local coordinates exist that relate displacement and elastic forces in a linear and coordinate independent manner. This would allow for a factorization of the matrix in a precomputation step or for multigrid methods without the need to update the whole matrix hierarchy in every step.

4.3.3. Accuracy

The investigations revealed weaker convergence rates in the cut-cell geometric multigrid solver for free-surface simulations with Dirichlet boundary conditions. The sub-grid Dirichlet boundaries for FV have to be developed further to improve the discretization order. If the compatibility between multigrid levels for these conditions is maintained, the convergence rates for the multigrid solver can be increased. In addition, the incorporation of moving geometry into fluid simulations with cut cells is an interesting option for further research. Moving geometry can induce interesting flow patterns, which require the analysis of methods for coupling solid geometry with the cut-cell representation in terms of interactivity.

A further interesting question is if the developed GPU-based and approximative methods for computer graphics can be applied in an engineering context to significantly speed up the development process in early stages. With a reduced number of elements and simplified models, such simulations can also draw conclusions on initial designs to guide the way to a pre-optimized shape. Furthermore, in engineering applications automatic optimization w.r.t. user-defined objective functions is important. One example is to change the geometry or the topology to save weight while simultaneously considering the stability of the part under load. The optimization loops are likely to be significantly accelerated by adopting the concepts presented in this thesis. It would be interesting to investigate if the simplified algorithms developed here might also be beneficial for optimization scenarios.

4.3.4. Further GPU extensions

The use of the *p-multigrid method* results in a significant acceleration of higher-order BB-form elements with a large number of degrees of freedom. Yet, with complex models this does not result in interactive rates as the computing times are clearly above 100 ms per time step. An extension to a massively parallel version is promising to result in speedups that allow for interactive simulations.

An obvious and challenging extension for both physical domains is the exploitation of multiple GPUs for interactive simulations. The extension should be able to use many processors on one graphics board, in a single machine or in a distributed environment. As the exchange of data between the different levels of the memory hierarchy will certainly be the main bottleneck, specialized methods and concepts must be developed to fully utilize

the available hardware and bandwidth between the hierarchies of memory. It has to be investigated how the algorithms scale with the use of multiple GPUs and whether this pays off in terms of higher number of elements for interactive scenarios. Furthermore, it might be worth coming up with a specific parallel programming language that takes the requirements of interactive simulations into account. Then, the algorithms could be implemented once and the distribution onto multiple, potentially heterogeneous processors could be done automatically.

4.3.5. Interactive simulation of coupled fluid and deformation effects

A combination of volumetric deformation and fluid simulation methods presented here opens up new possibilities for an interactive coupling of both physical domains. Fluid flow induces forces on elastic geometry which then deforms accordingly. Likewise, the motion of the elastic body affects fluid flow through its displacement. Therefore, further investigations are required if higher-order FEs for deformation simulation can be combined with the cut-cell approach for fluid simulation in order to improve the accuracy of a fluid-solid coupling in an interactive scenario.

A. Notation

A.1. Simulation of dynamic volumetric deformation

$\mathbf{A}$	System matrix
$\mathbf{b}$	Force densities
B_{ijkl} or B_I	Bernstein polynomial
$\mathbf{B}_{\mathbb{T}}$	Differential basis function matrix
b_{ijkl}	Value at a FE node in BB-form
$\mathbf{C}$	Cauchy-Green deformation tensor
$\mathrm{d}\mathbf{x}$ and $\mathrm{d}\mathbf{X}$	Infinitesimal line segments in simulation domain Ω and deformed domain Ω'
D_{ijkl}	Elasticity tensor
$\mathbf{D}$	Elasticity tensor in Voigt notation
E	Young's modulus
$\mathcal{E}$	Set of edges
e_E	One-dimensional engineering strain
$\boldsymbol{\varepsilon}$	Strain tensor in Voigt notation
ε and ε_C	Cauchy strain
ε_E	Infinitesimal engineering strain
ε_G	Green strain
ε_{CR}	Corotational strain
$\mathcal{F}$	Set of faces
$\mathbf{F}$	Deformation gradient
$\mathbf{f}^{\text{ext}}$	External forces
$\mathbf{f}^{\text{el}}$	Elastic forces
$\mathbf{f}^{\text{corot}}$	Corotational elastic forces
$\mathbf{f}^{0}$	Constant corotational elastic forces
$\mathbf{g}$	Gravitation forces
Γ	Boundary of the simulation domain Ω
Γ_D and Γ_N	Dirichlet and Neumann boundary conditions
$G(I,J)$	Function of binomial coefficients
$\mathbb{I}$	3×3 identity matrix
$\mathbf{I}_l^{l+1}$	Restriction operator
$\mathbf{I}_{l+1}^{l}$	Prolongation operator
K	Bulk modulus
$\mathbf{K}$	Global stiffness matrix
$\mathbf{K}_{\mathbb{T}}$	Element stiffness matrix
$\mathbf{K}_{\mathbb{T}}^{p}$	Element stiffness matrix for a degree p FE discretization
${}^{0}\mathbf{K}_{\mathbb{T}}$	Initial element stiffness matrix
λ_i	Barycentric coordinates
λ_L	1st Lamé coefficient
μ_L	Shear modulus
Ω	Computational domain
Ω'	Deformed computational domain
ω	Infinitesimal rotation tensor

p	Polynomial degree
$\mathbf{P}$	Preconditioner matrix
$\mathbf{p}_i$	Vertices of a tetrahedron
φ	Deformation map relating simulation domain Ω to deformed domain Ω'
m	Number of degrees of freedom for the whole mesh
$\mathbf{M}$	Global mass matrix
$\mathbf{M}_{\mathbb{T}}$	Element mass matrix
$\mathbf{M}_{\mathbb{T}}^{p}$	Element mass matrix for a degree p FE discretization
$\mathbf{M}_L$	Lumped mass matrix
n	Number of degrees of freedom per element
∇	Differential operator *Nabla* representing gradient, divergence or rotation
N_i	Polynomial basis function
$\mathbf{N}_{\mathbb{T}}$	Matrix of basis function for an element $\mathbb{T}$
$\mathbf{n}$	Normal vector
ν	Poisson ratio
ρ	Density
$\mathbf{r}$	Residual vector
$\mathbf{R}$	Rotation matrix
$\mathbf{S}$	Stretch tensor obtained by polar decomposition
$\triangle$	Tetrahedral mesh
$\mathcal{T}$	Set of tetrahedra
σ	Stress tensor
$\boldsymbol{\sigma}$	Stress tensor in Voigt notation
Δt	Time step
$\mathbb{T}$	Tetrahedral element
$\mathbf{t}$	Traction
$\mathcal{V}$	Set of vertices
$\mathbf{u}(\mathbf{x})$	Displacement field
$\ddot{\mathbf{u}}$	Acceleration of displacement field
$\mathbf{w}$	Test functions
χ	Mapping function relating local to global indices in the tetrahedral mesh

A.2. Fluid simulation

$\mathbf{a}$	Arbitrary vector field
A_f, $A_{i\pm\frac{1}{2}jk}$, $A_{ij\pm\frac{1}{2}k}$ $A_{ijk\pm\frac{1}{2}}$	Face fractions
A^k	Array representation of a five-point or seven-point Laplace discretization
A_d^{proj}	Projected area of the Dirichlet boundary
$\mathbf{b}$	Force densities
C_{ijk}	Grid cell in the computational mesh
Δ	Laplace operator
ε	Strain
$\dot{\varepsilon}$	Strain rate
η	Kinematic viscosity
$\mathbf{f}$	External force
Γ	Boundary of the computational domain and interface between liquid and air phase
Γ_D and Γ_N	Dirichlet and Neumann boundary conditions
Γ_P	Closest point on the free surfaces w.r.t. a point in the simulation domain
$\mathbb{I}$	3×3 identity matrix
$\mathbf{I}_h^H$	Restriction operator
$\mathbf{I}_H^h$	Restriction operator
κ	Condition number of a matrix
$\mathbf{L}$	Matrix of the linear system for the pressure correction
Ω	Computational domain
φ	Map relating current to initial state
ϕ and ϕ_{ijk}	Level set function and samples
Φ	Arbitrary scalar field on a grid
m	Number of non-zero entries in the sparse matrix
μ	Shear viscosity
n	Number of degrees of freedom
∇	Differential operator *Nabla* representing gradient, divergence or rotation
n_x, n_y, n_z	Number of grid cells in x, y and z-direction
$\mathbf{n}$ and $\mathbf{n}_f$	Normal vector and face normal of a grid cell
p and p_{ijk}	Pressure field and sample
p^G	Ghost value for a pressure sample outside the simulation domain
$\mathbf{r}$	Residual vector
ρ	Density
σ	Stress tensor
τ	Deviatoric or viscous stress tensor
θ	Fractional distance to the free surface
u and $u_{i\pm\frac{1}{2}jk}$	Velocity component and sample in x-direction
v and $v_{ij\pm\frac{1}{2}k}$	Velocity component and sample in y-direction
$\mathbf{v}(\mathbf{x})$	Velocity field
$\mathbf{v}^t$	Velocity field at time step t
$\mathbf{v}^f$	Velocity field after applying forces
$\mathbf{v}^a$	Velocity field after applying advection term
$\mathbf{v}^*$	Velocity field before pressure correction
V_{ijk}	Volume of a grid cell
w and $w_{ijk\pm\frac{1}{2}}$	Velocity component and sample in z-direction
ζ	Second coefficient of viscosity
Δx	Mesh spacing
$\mathbf{X}^*$	Endpoint of advection

B. List of abbreviations

BB-form	Bernstein-Bézier-form
BCSR	Block compressed sparse row
BFECC	Back and forth error compensation and correction
BLAS	Basic linear algebra subprograms
CFD	Computational fluid dynamics
CFL condition	Courant-Friedrichs-Lewy condition
CUDA	Compute unified device architecture
CG	Conjugate gradient algorithm
CSR	Compressed sparse row
DEC	Discrete exterior calculus
DGM	Discontinuous Galerkin method
FD / FDM	Finite difference / Finite difference method
FE / FEM	Finite element / Finite element method
FLIP	Fluid implicit particle
FV / FVM	Finite volume / Finite volume method
GFM	Ghost fluid method
GPU	Graphics processing unit
GPGPU	General purpose-computing on graphics processing units
ICCG	Incomplete Cholesky preconditioned conjugate gradient
MGPCG	Multigrid preconditioned conjugate gradient solver
MCG	Merged conjugate gradient
ODE	Ordinary differential equation
PCG	(Diagonal) Preconditioned conjugate gradient algorithm
PDE	Partial differential equation
PIC	Particle in cell
PMG	p-multigrid
SIMD	Single instruction multiple data
SIMT	Single instruction multiple threads
SPH	Smoothed particle hydrodynamics
SpMV	Sparse matrix-vector multiplication
SpMM	Sparse matrix-matrix multiplication
SM	Streaming multiprocessor
StVK	St. Venant-Kirchhoff (elastic material)
SVD	Singular value decomposition

C. Publications and talks

C.1. Authored and co-authored publications

[BDW*10] BINOTTO A., DANIEL C. G., WEBER D., KUIJPER A., STORK A., PEREIRA C. E., FELLNER D. W.: Iterative SLE solvers over a CPU-GPU platform. In *Proceedings 2010 12th IEEE International Conference on High Performance Computing and Communications* (2010), pp. 305–313. doi:10.1109/HPCC.2010.40

[WKS*11] WEBER D., KALBE T., STORK A., FELLNER D. W., GOESELE M.: Interactive deformable models with quadratic bases in Bernstein-Bézier-form. *The Visual Computer 27* (2011), 473–483. doi:10.1007/s00371-011-0579-6

[WPnSSF11] WEBER D., PEÑA SERNA S., STORK A., FELLNER D. W.: Rapid CFD for the early conceptual design phase. In *The Integration of CFD into the Product Development Process* (2011), International Association for the Engineering Analysis Community (NAFEMS), NAFEMS, Glasgow, p. 9

[SSW*12] STORK A., SEVILMIS N., WEBER D., GORECKY D., STAHL C., LOSKYLL M., MICHEL F.: Enabling virtual assembly training in and beyond the automotive industry. In *Proceedings of the VSMM 2012* (2012), International Society on Virtual Systems and MultiMedia, IEEE Computer Society, Los Alamitos, Calif., pp. 347–352. doi:10.1109/VSMM.2012.6365944

[WBS*13] WEBER D., BENDER J., SCHNOES M., STORK A., FELLNER D. W.: Efficient GPU data structures and methods to solve sparse linear systems in dynamics applications. *Computer Graphics Forum 32*, 1 (2013), 16–26. doi:10.1111/j.1467-8659.2012.03227.x

[BWD13] BENDER J., WEBER D., DIZIOL R.: Fast and stable cloth simulation based on multi-resolution shape matching. *Computers & Graphics 37*, 8 (2013), 945 – 954. doi:10.1016/j.cag.2013.08.003

[WMRA*14] WEBER D., MUELLER-ROEMER J., ALTENHOFEN C., STORK A., FELLNER D. W.: A p-multigrid algorithm using cubic finite elements for efficient deformation simulation. In *Virtual Reality Interactions and Physical Simulations (VRIPhys)* (2014), pp. 49–58. doi:10.2312/vriphys.20141223

[BKCW14] BENDER J., KOSCHIER D., CHARRIER P., WEBER D.: Position-based simulation of continuous materials. *Computers & Graphics 44* (2014), 1 – 10. doi:10.1016/j.cag.2014.07.004

[ABH*14] ADACHI S., BRANDSTÄTT P., HERGET W., LEISTNER P., LANDERSHEIM V., WEBER D., MUELLER ROEMER J.: Wind tunnel test and CFD/CAA analysis on a scaled model of a nose landing gear. In *Greener Aviation. Clean Sky Breakthroughs and Worldwide Status* (2014), Council of European Aerospace Societies (CEAS), p. 8

[WMRSF15] WEBER D., MUELLER-ROEMER J., STORK A., FELLNER D. W.: A cut-cell geometric multigrid poisson solver for fluid simulation. *Computer Graphics Forum (Eurographics proceedings) 34*, 2 (2015), 481–491. doi:10.1111/cgf.12577

[WMRA*15] WEBER D., MUELLER-ROEMER J., ALTENHOFEN C., STORK A., FELLNER D. W.: Deformation simulation using cubic finite elements and efficient *p*-multigrid methods. *Computers & Graphics 53, Part B* (2015), 185 – 195. doi:10.1016/j.cag.2015.06.010

C.2. Talks

1. Eurographics 2015: A cut-cell geometric multigrid poisson solver for fluid simulation. Zürich, Switzerland, 2015.
2. Eurographics 2014: Efficient GPU data structures and methods to solve sparse linear systems in dynamics applications. Invitation to Eurographics (Computer Graphics Forum publication). Strasbourg, France, 2014.
3. 12th Workshop on Virtual Reality Interaction and Physical Simulation: A p-multigrid algorithm using cubic finite Elements for efficient deformation simulation. Bremen, Germany, 2014.
4. SIGGRAPH, Talk in a birds of a feather session: Interactivity in simulation applications. Los Angeles, USA, 2012.
5. VDI-Vortragsreihe: Simulierte Realität: Interaktive Simulationen. Darmstadt, Germany, 2012
6. Computer Graphics International: Interactive deformable models with quadratic bases in Bernstein-Bézier-form. Ottawa, Canada, 2011
7. ANSYS conference & CADFEM Users' Meeting: Potenzial interaktiver Strömungssimulationen. Leipzig, Germany, 2009
8. ANSYS conference & CADFEM Users' Meeting: Echtzeitsimulation: Stand und Möglichkeiten. Darmstadt, Germany, 2008
9. NAFEMS Seminar: Die Integration von Strömungsberechnungen (CFD) in den Produktentwicklungsprozess: Rapid CFD for the early Conceptual Design Phase. Wiesbaden, Germany, 2011.

D. Supervising Activities

The following list summarizes the student bachelor, diploma and master thesis supervised by the author. The results of these works were partially used as an input into the thesis.

D.1. Diploma and Master Theses

1. Lassmann, A.: Interaktive physikalisch-basierte Simulation auf verteilten Systemen. Master Thesis, Technische Universität Darmstadt (Visual Computing), 2015.

D.2. Bachelor Theses

1. Schuwirth, F.: Effiziente Simulation von Masse-Feder-Systemen. Bachelor Thesis, Technische Universität Darmstadt (Computer Science), 2015.
2. Daun, K.: Collision Handling between Rigid and Deformable Bodies with Continuous Penalty Forces. Bachelor Thesis, Technische Universität Darmstadt (Computational Engineering), 2014.
3. Grasser, T.: Energy-Preserving Integrators for Fluid Animation with Discrete Exterior Calculus on Two-Dimensional Meshes. Bachelor Thesis, Technische Universität Darmstadt (Mathematics with Computer Science), 2012.
4. Müller, C.: Splines auf Tetraederpartitionen für physikalisch basierte Deformationssimulation. Bachelor Thesis, Technische Universität Darmstadt (Computational Engineering), 2011.
5. Räsch, S.: Two-dimensional Circulation-preserving Fluid Simulation with Discrete Exterior Calculus. Bachelor Thesis, Technische Universität Darmstadt (Mathematics with Computer Science), 2011.
6. Gast, J.: A Geometric Multigrid Method for Simulating Deformable Models on Unstructured, Non-nested Mesh Hierarchies. Bachelor Thesis, Technische Universität Darmstadt (Computational Engineering), 2010.
7. Keller, K.: Efficient Multigrid Methods for Realtime Simulation. Bachelor Thesis, Technische Universität Darmstadt (Computational Engineering), 2010.

E. Curriculum Vitae

Personal Data

Name	Daniel Weber
Family status	married, one daughter
Nationality	German

Education

02/2008	Graduation in computer science, Technische Universität Darmstadt
2000 – 2008	Study at Technische Universität Darmstadt
1999	Abitur, Georg-Büchner-Schule, Darmstadt

Work Experience

Since 04/2014	Deputy head of department *Interactive Engineering Technologies*
Since 10/2013	Head of group *Interactive Simulation*
Since 03/2008	Researcher, Fraunhofer Institute of Computer Graphics Research IGD, department *Interactive Engineering Technologies*, Focus: Interactive Simulation.

Bibliography

[ABH*14] ADACHI S., BRANDSTÄTT P., HERGET W., LEISTNER P., LANDERSHEIM V., WEBER D., MUELLER ROEMER J.: Wind tunnel test and CFD/CAA analysis on a scaled model of a nose landing gear. In *Greener Aviation. Clean Sky Breakthroughs and Worldwide Status* (2014), Council of European Aerospace Societies (CEAS), p. 8. 165

[ACF11] ALLARD J., COURTECUISSE H., FAURE F.: Implicit FEM solver on GPU for interactive deformation simulation. In *GPU Computing Gems Jade Edition*. NVIDIA/Elsevier, Sept. 2011, ch. 21, pp. 281–294. doi:10.1016/B978-0-12-385963-1.00021-6. 41, 66, 90

[ATW13] ANDO R., THÜREY N., WOJTAN C.: Highly adaptive liquid simulations on tetrahedral meshes. *ACM Trans. Graph. 32*, 4 (July 2013), 103:1–103:10. doi:10.1145/2461912.2461982. 116

[ATW15] ANDO R., THÜREY N., WOJTAN C.: A dimension-reduced pressure solver for liquid simulations. *Computer Graphics Forum 34*, 2 (2015), 473–480. doi:10.1111/cgf.12576. 116

[BB09a] BASKARAN M. M., BORDAWEKAR R.: *Optimizing Sparse Matrix-Vector Multiplications on GPUs*. Tech. rep., IBM Reserach Report, RC24704 (W0812-047), 2009. 40, 79, 90

[BB09b] BROCHU T., BRIDSON R.: Robust topological operations for dynamic explicit surfaces. *SIAM J. Sci. Comput. 31*, 4 (June 2009), 2472–2493. doi:10.1137/080737617. 114

[BB12] BOYD L., BRIDSON R.: MultiFLIP for energetic two-phase fluid simulation. *ACM Trans. Graph. 31*, 2 (Apr. 2012), 16:1–16:12. doi:10.1145/2159516.2159522. 114

[BBB07] BATTY C., BERTAILS F., BRIDSON R.: A fast variational framework for accurate solid-fluid coupling. *ACM Trans. Graph. 26*, 3 (July 2007). doi:10.1145/1276377.1276502. 117, 118, 122

[BBB10] BROCHU T., BATTY C., BRIDSON R.: Matching fluid simulation elements to surface geometry and topology. *ACM Trans. Graph. 29*, 4 (July 2010), 47:1–47:9. doi:10.1145/1778765.1778784. 114

[BBC*94] BARRETT R., BERRY M. W., CHAN T. F., DEMMEL J., DONATO J., DONGARRA J., EIJKHOUT V., POZO R., ROMINE C., VAN DER VORST H.: *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, vol. 43. Siam, 1994. URL: http://www.netlib.org/templates/templates.pdf. 31

[BC14] BARGTEIL A. W., COHEN E.: Animation of deformable bodies with quadratic Bézier finite elements. *ACM Trans. Graph. 33*, 3 (June 2014), 27:1–27:10. doi:10.1145/2567943. 38, 56, 59, 63

[BCL09] BUATOIS L., CAUMON G., LEVY B.: Concurrent number cruncher: a GPU implementation of a general sparse linear solver. *Int. J. Parallel Emerg. Distrib. Syst. 24* (June 2009), 205–223. doi:10.1080/17445760802337010. 40, 79, 82, 90

[BDW*10] BINOTTO A., DANIEL C. G., WEBER D., KUIJPER A., STORK A., PEREIRA C. E., FELLNER D. W.: Iterative SLE solvers over a CPU-GPU platform. In *Proceedings 2010 12th IEEE International Conference on High Performance Computing and Communications* (2010),

pp. 305–313. doi:10.1109/HPCC.2010.40. 165

[BETC12] BENDER J., ERLEBEN K., TRINKLE J., COUMANS E.: Interactive simulation of rigid body dynamics in computer graphics. In *Eurographics - State of the Art Reports* (Cagliari, Sardinia, Italy, 2012), Cani M.-P., Ganovelli F., (Eds.), Eurographics Association, pp. 95–134. doi:10.2312/conf/EG2012/stars/095-134. 85

[BFGS03] BOLZ J., FARMER I., GRINSPUN E., SCHRÖDER P.: Sparse matrix solvers on the GPU: Conjugate gradients and multigrid. *ACM Trans. Graph. 22* (2003), 917–924. doi:10.1145/1201775.882364. 40

[BG08] BELL N., GARLAND M.: *Efficient Sparse Matrix-Vector Multiplication on CUDA*. NVIDIA Technical Report NVR-2008-004, NVIDIA Corporation, Dec. 2008. 40, 79

[BG09] BELL N., GARLAND M.: Implementing sparse matrix-vector multiplication on throughput-oriented processors. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis* (New York, NY, USA, 2009), SC '09, ACM, pp. 18:1–18:11. doi:10.1145/1654059.1654078. 40, 77, 79, 80, 86, 90

[BG10] BELL N., GARLAND M.: CUSP: Generic parallel algorithms for sparse matrix and graph computations, 2010. URL: http://cusp-library.googlecode.com. 40, 86

[BGOS06] BARGTEIL A. W., GOKTEKIN T. G., O'BRIEN J. F., STRAIN J. A.: A semi-Lagrangian contouring method for fluid simulation. *ACM Trans. Graph. 25*, 1 (Jan. 2006), 19–38. doi:10.1145/1122501.1122503. 114

[BHM00] BRIGGS W. L., HENSON V. E., MCCORMICK S. F.: *A multigrid tutorial (2nd ed.)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000. doi:10.1137/1.9780898719505. 39, 60, 118

[BHW13] BOJSEN-HANSEN M., WOJTAN C.: Liquid surface tracking with error compensation. *ACM Trans. Graph. 32*, 4 (July 2013), 68:1–68:13. doi:10.1145/2461912.2461991. 114

[BJ05] BARBIC J., JAMES D. L.: Real-time subspace integration for St. Venant-Kirchhoff deformable models. *ACM Trans. Graph. 24*, 3 (July 2005), 982–990. doi:10.1145/1073204.1073300. 7, 37

[BKCW14] BENDER J., KOSCHIER D., CHARRIER P., WEBER D.: Position-based simulation of continuous materials. *Computers & Graphics 44* (2014), 1 – 10. doi:10.1016/j.cag.2014.07.004. 165

[BMF07] BRIDSON R., MÜLLER-FISCHER M.: Fluid simulation: Siggraph 2007 course notes. In *ACM SIGGRAPH 2007 Courses* (New York, NY, USA, 2007), SIGGRAPH '07, ACM, pp. 1–81. doi:10.1145/1281500.1281681. 110, 111

[BMO*12] BENDER J., MÜLLER M., OTADUY M. A., TESCHNER M., MACKLIN M.: A Survey on Position-Based Simulation Methods in Computer Graphics. *Computer Graphics Forum* (2012). doi:10.1111/cgf.12346. 34

[BN98] BRO-NIELSEN M.: Finite element modeling in surgery simulation. *Proceedings of the IEEE 86*, 3 (Mar 1998), 490–503. doi:10.1109/5.662874. 35

[Bri08] BRIDSON R.: *Fluid Simulation For Computer Graphics*. A K Peters Series. A K Peters, Limited, 2008. URL: http://books.google.de/books?id=gFI8y87VCZ8C. 104, 105, 109, 112, 133, 134, 138

[BS06] BENDER J., SCHMITT A.: Fast dynamic simulation of multi-body systems using impulses. In *Virtual Reality Interactions and Physical Simulations (VRIPhys)* (Madrid (Spain), Nov. 2006),

pp. 81–90. doi:10.2312/PE/vriphys/vriphys06/081-090. 85

[BW98] BARAFF D., WITKIN A.: Large steps in cloth simulation. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1998), SIGGRAPH '98, ACM, pp. 43–54. doi:10.1145/280814.280821. 30, 32, 35, 60, 64, 81, 82, 84

[BW08] BONET J., WOOD R. D.: *Nonlinear Continuum Mechanics for FEA*. Cambridge University Press, NY, 2008. doi:10.1017/CBO9780511755446. 10, 21, 34

[BWD13] BENDER J., WEBER D., DIZIOL R.: Fast and stable cloth simulation based on multi-resolution shape matching. *Computers & Graphics 37*, 8 (2013), 945 – 954. doi:10.1016/j.cag.2013.08.003. 165

[BXH10] BATTY C., XENOS S., HOUSTON B.: Tetrahedral embedded boundary methods for accurate and flexible adaptive fluids. *Computer Graphics Forum 29*, 2 (2010), 695–704. doi:10.1111/j.1467-8659.2009.01639.x. 116

[BZ11] BARBIC J., ZHAO Y.: Real-time large-deformation substructuring. *ACM Trans. Graph. 30*, 4 (July 2011), 91:1–91:8. doi:10.1145/2010324.1964986. 36

[CA09] COURTECUISSE H., ALLARD J.: Parallel dense Gauss-Seidel algorithm on many-core processors. In *Proceedings of the 2009 11th IEEE International Conference on High Performance Computing and Communications* (Washington, DC, USA, 2009), IEEE Computer Society, pp. 139–147. doi:10.1109/HPCC.2009.51. 41

[CFL*07] CHENTANEZ N., FELDMAN B. E., LABELLE F., O'BRIEN J. F., SHEWCHUK J. R.: Liquid simulation on lattice-based tetrahedral meshes. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics symposium on Computer animation* (Aire-la-Ville, Switzerland, Switzerland, 2007), SCA '07, Eurographics Association, pp. 219–228. URL: http://dl.acm.org/citation.cfm?id=1272690.1272720. 115

[CGA] CGAL, Computational Geometry Algorithms Library. URL: http://www.cgal.org. 21, 43

[Cho67] CHORIN A. J.: A numerical method for solving incompressible viscous flow problems. *J. Comput. Phys. 2*, 1 (1967), 12 – 26. doi:10.1016/0021-9991(67)90037-X. 107

[CIR52] COURANT R., ISAACSON E., REES M.: On the solution of nonlinear hyperbolic differential equations by finite differences. *Communications on Pure and Applied Mathematics 5*, 3 (1952), 243–255. doi:10.1002/cpa.3160050303. 113

[CK02] CHOI K.-J., KO H.-S.: Stable but responsive cloth. *ACM Trans. Graph. 21*, 3 (2002), 604–611. doi:10.1145/566654.566624. 86

[CLT07] CRANE K., LLAMAS I., TARIQ S.: *Real Time Simulation and Rendering of 3D Fluids*. Addison-Wesley, 2007, ch. 30. URL: http://http.developer.nvidia.com/GPUGems3/gpugems3_ch30.html. 119, 129

[CM11a] CHENTANEZ N., MÜLLER M.: A multigrid fluid pressure solver handling separating solid boundary conditions. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (New York, NY, USA, 2011), SCA '11, ACM, pp. 83–90. doi:10.1145/2019406.2019418. 118, 133, 139, 146

[CM11b] CHENTANEZ N., MÜLLER M.: Real-time Eulerian water simulation using a restricted tall cell grid. *ACM Trans. Graph. 30*, 4 (July 2011), 82:1–82:10. doi:10.1145/2010324.1964977. 118, 119, 133, 135, 140

[CP03] CLINE M., PAI D.: Post-stabilization for rigid body simulation with contact and constraints. *Proceedings of IEEE International Conference on Robotics and Automation 3* (September 2003), 3744 – 3751. doi:10.1109/ROBOT.2003.1242171. 85

[CPSS10] CHAO I., PINKALL U., SANAN P., SCHRÖDER P.: A simple geometric model for elastic deformations. *ACM Trans. Graph. 29*, 4 (2010), 38:1–38:6. doi:10.1145/1778765.1778775. 36

[CRM91] CARD S. K., ROBERTSON G. G., MACKINLAY J. D.: The information visualizer, an information workspace. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 1991), CHI '91, ACM, pp. 181–186. doi:10.1145/108844.108874. 1, 7

[CTG10] COHEN J. M., TARIQ S., GREEN S.: Interactive fluid-particle simulation using translating Eulerian grids. In *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2010), I3D '10, ACM, pp. 15–22. doi:10.1145/1730804.1730807. 119

[Dav06] DAVIS T. A.: *Direct Methods for Sparse Linear Systems (Fundamentals of Algorithms 2)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2006. doi:10.1137/1.9780898718881. 31

[DBG14] DA F., BATTY C., GRINSPUN E.: Multimaterial mesh-based surface tracking. *ACM Trans. on Graphics (SIGGRAPH 2014)* (2014). doi:10.1145/2601097.2601146. 115

[DDCB01] DEBUNNE G., DESBRUN M., CANI M.-P., BARR A. H.: Dynamic real-time deformations using space & time adaptive sampling. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 2001), SIGGRAPH '01, ACM, pp. 31–36. doi:10.1145/383259.383262. 30

[DGW11a] DICK C., GEORGII J., WESTERMANN R.: A hexahedral multigrid approach for simulating cuts in deformable objects. *Visualization and Computer Graphics, IEEE Transactions on 17*, 11 (2011), 1663–1675. doi:10.1109/TVCG.2010.268. 39, 73

[DGW11b] DICK C., GEORGII J., WESTERMANN R.: A real-time multigrid finite hexahedra method for elasticity simulation using CUDA. *Simulation Modelling Practice and Theory 19*, 2 (2011), 801–816. doi:10.1016/j.simpat.2010.11.005. 39, 41, 73

[EB14] EDWARDS E., BRIDSON R.: Detailed water with coarse grids: Combining surface meshes and adaptive discontinuous Galerkin. *ACM Trans. Graph. 33*, 4 (July 2014), 136:1–136:9. doi:10.1145/2601097.2601167. 115

[EF02] ENRIGHT D., FEDKIW R.: Robust treatment of interfaces for fluid flows and computer graphics. In *9th International Conference on Hyperbolic Systems on Hyperbolic Problems Theory* (2002). doi:10.1007/978-3-642-55711-8_13. 114

[EFFM02] ENRIGHT D., FEDKIW R., FERZIGER J., MITCHELL I.: A hybrid particle level set method for improved interface capturing. *J. Comput. Phys. 183*, 1 (Nov. 2002), 83–116. doi:10.1006/jcph.2002.7166. 114

[EKS03] ETZMUSS O., KECKEISEN M., STRASSER W.: A fast finite element solution for cloth modelling. In *Proceedings of the 11th Pacific Conference on Computer Graphics and Applications* (Washington, DC, USA, 2003), PG '03, IEEE Computer Society, pp. 244–. URL: http://dl.acm.org/citation.cfm?id=946250.946946. 46

[EMF02] ENRIGHT D., MARSCHNER S., FEDKIW R.: Animation and rendering of complex water surfaces. *ACM Trans. Graph. 21*, 3 (July 2002), 736–744. doi:10.1145/566654.566645. 114

[ENGF03] ENRIGHT D., NGUYEN D., GIBOU F., FEDKIW R.: Using the particle level set method and a second order accurate pressure boundary condition for free surface flows. In *In Proc. 4th ASME-JSME Joint Fluids Eng. Conf.* (2003), pp. 2003–45144. doi:10.1115/FEDSM2003-45144. 135

[EQYF13] ENGLISH R. E., QIU L., YU Y., FEDKIW R.: Chimera grids for water simulation. In *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (New York, NY, USA, 2013), SCA '13, ACM, pp. 85–94. doi:10.1145/2485895.2485897. 116

[ETK*07] ELCOTT S., TONG Y., KANSO E., SCHRÖDER P., DESBRUN M.: Stable, circulation-preserving, simplicial fluids. *ACM Trans. Graph. 26*, 1 (Jan. 2007). doi:10.1145/1189762.1189766. 116

[FAMO99] FEDKIW R. P., ASLAM T., MERRIMAN B., OSHER S.: A non-oscillatory Eulerian approach to interfaces in multimaterial flows (the ghost fluid method). *J. Comput. Phys. 152*, 2 (July 1999), 457–492. doi:10.1006/jcph.1999.6236. 117

[Far02] FARIN G.: *Curves and Surfaces for CAGD: A Practical Guide*, 5th ed. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002. 34, 43, 61, 64

[FGBP11] FAURE F., GILLES B., BOUSQUET G., PAI D. K.: Sparse meshless models of complex deformable solids. *ACM Trans. Graph. 30*, 4 (July 2011), 73:1–73:10. doi:10.1145/2010324.1964968. 37

[FLLP13] FAN Y., LITVEN J., LEVIN D. I. W., PAI D. K.: Eulerian-on-Lagrangian simulation. *ACM Trans. Graph. 32*, 3 (July 2013), 22:1–22:9. doi:10.1145/2487228.2487230. 37, 100

[FLP14] FAN Y., LITVEN J., PAI D. K.: Active volumetric musculoskeletal systems. *ACM Trans. Graph. 33*, 4 (July 2014), 152:1–152:9. doi:10.1145/2601097.2601215. 38

[FM96] FOSTER N., METAXAS D.: Realistic animation of liquids. *Graphical models and image processing 58*, 5 (1996), 471–483. doi:10.1006/gmip.1996.0039. 113, 114, 133

[FM97] FOSTER N., METAXAS D.: Modeling the motion of a hot, turbulent gas. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques* (1997), ACM Press/Addison-Wesley Publishing Co., pp. 181–188. doi:10.1145/258734.258838. 113

[FOK05] FELDMAN B. E., O'BRIEN J. F., KLINGNER B. M.: Animating gases with hybrid meshes. *ACM Trans. Graph. 24*, 3 (July 2005), 904–909. doi:10.1145/1073204.1073281. 115

[FP02] FERZIGER J. H., PERIC M.: *Computational Methods for Fluid Dynamics*, vol. 3. Springer-Verlag Berlin Heidelberg, 2002. doi:10.1007/978-3-642-56026-2. 104, 112

[FSH11] FIERZ B., SPILLMANN J., HARDERS M.: Element-wise mixed implicit-explicit integration for stable dynamic simulation of deformable objects. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (New York, NY, USA, 2011), SCA '11, ACM, pp. 257–266. doi:10.1145/2019406.2019440. 36

[FSJ01] FEDKIW R., STAM J., JENSEN H. W.: Visual simulation of smoke. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2001), SIGGRAPH '01, ACM, pp. 15–22. doi:10.1145/383259.383260. 104, 113

[Fun94] FUNG Y. C.: *A first course in continuum mechanics, third edn*. Prentice Hall, Englewood Cliffs, N.J., 1994. doi:10.1017/CBO9780511619571. 10, 17, 34

[FWD14] FERSTL F., WESTERMANN R., DICK C.: Large-scale liquid simulation on adaptive hexahedral grids. *Visualization and Computer Graphics, IEEE Transactions on 20*, 10 (Oct 2014), 1405–

1417. doi:10.1109/TVCG.2014.2307873. 118, 133, 146

[GFCK02] GIBOU F., FEDKIW R. P., CHENG L.-T., KANG M.: A second-order-accurate symmetric discretization of the Poisson equation on irregular domains. *J. Comput. Phys. 176*, 1 (Feb. 2002), 205–227. doi:10.1006/jcph.2001.6977. 111, 117

[GHSW07] GROSS D., HAUGER W., SCHRÖDER J., WALL W.: *Technische Mechanik 2 - Elastostatik*. Springer-Verlag, 2007. doi:10.1007/978-3-642-40966-0. 66

[GJ*10] GUENNEBAUD G., JACOB B., ET AL.: Eigen v3, 2010. URL: http://eigen.tuxfamily.org. 31, 32, 67

[GKS02] GRINSPUN E., KRYSL P., SCHRÖDER P.: CHARMS: A simple framework for adaptive simulation. *ACM Trans. Graph. 21*, 3 (July 2002), 281–290. doi:10.1145/566654.566578. 37

[GM97] GIBSON S. F. F., MIRTICH B.: *A Survey of Deformable Modeling in Computer Graphics*. Tech. Rep. TR97-19, MERL - Mitsubishi Electric Research Laboratories, Cambridge, MA 02139, Nov. 1997. URL: http://www.merl.com/publications/TR97-19/. 34

[GW06] GEORGII J., WESTERMANN R.: A multigrid framework for real-time simulation of deformable bodies. *Comput. Graph. 30*, 3 (June 2006), 408–415. doi:10.1016/j.cag.2006.02.016. 39, 69

[GW08] GEORGII J., WESTERMANN R.: Corotated finite elements made fast and stable. In *VRIPHYS* (Grenoble, France, 2008), pp. 11–19. doi:10.2312/PE/vriphys/vriphys08/011-019. 36, 39, 43, 47, 48, 50, 69

[Har04] HARRIS M.: *Fast fluid dynamics simulation on the GPU*. Addison Wesley, 2004, ch. 37, pp. 637–665. URL: http://http.developer.nvidia.com/GPUGems/gpugems_ch38.html. 119, 129

[HBW03] HOUSTON B., BOND C., WIEBE M.: A unified approach for modeling complex occlusions in fluid simulations. In *ACM SIGGRAPH 2003 Sketches & Applications* (New York, NY, USA, 2003), SIGGRAPH '03, ACM, pp. 1–1. doi:10.1145/965400.965561. 138

[HES03] HAUTH M., ETZMUSS O., STRASSER W.: Analysis of numerical methods for the simulation of deformable models. *The Visual Computer 19*, 7-8 (2003), 581–600. doi:10.1007/s00371-003-0206-2. 36

[HLSO12] HECHT F., LEE Y. J., SHEWCHUK J. R., O'BRIEN J. F.: Updated sparse Cholesky factors for corotational elastodynamics. *ACM Transactions on Graphics 31*, 5 (Oct. 2012), 123:1–13. Presented at SIGGRAPH 2012. doi:10.1145/2231816.2231821. 31

[HS04] HAUTH M., STRASSER W.: Corotational simulation of deformable solids. In *Proceedings of WSCG 2004* (2004), UNION Agency-Science Press, pp. 137–145. 15, 27, 35, 46, 53

[HW65] HARLOW F. H., WELCH J. E.: Numerical calculation of time-dependent viscous incompressible flow of fluid with a free surface. *The Physics of Fluids 8*, 12 (1965), 2182–2189. 105, 113, 114

[IGLF06] IRVING G., GUENDELMAN E., LOSASSO F., FEDKIW R.: Efficient simulation of large bodies of water by coupling two and three dimensional techniques. *ACM Trans. Graph. 25*, 3 (July 2006), 805–811. doi:10.1145/1141911.1141959. 119

[IOS*14] IHMSEN M., ORTHMANN J., SOLENTHALER B., KOLB A., TESCHNER M.: SPH fluids in computer graphics. In *Eurographics 2014 - State of the Art Reports* (2014), Lefebvre S., Spagnuolo M., (Eds.), The Eurographics Association. doi:10.2312/egst.20141034. 116

[ITF04] IRVING G., TERAN J., FEDKIW R.: Invertible finite elements for robust simulation of large deformation. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation* (Aire-la-Ville, Switzerland, Switzerland, 2004), SCA '04, Eurographics Association, pp. 131–140. doi:10.1145/1028523.1028541. 29, 35, 47

[JC98] JOHANSEN H., COLELLA P.: A cartesian grid embedded boundary method for Poisson's equation on irregular domains. *J. Comput. Phys. 147*, 1 (Nov. 1998), 60–85. doi:10.1006/jcph.1998.5965. 117, 118

[JP99] JAMES D. L., PAI D. K.: Artdefo: Accurate real time deformable objects. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1999), SIGGRAPH '99, ACM Press/Addison-Wesley Publishing Co., pp. 65–72. doi:10.1145/311535.311542. 36

[KFCO06] KLINGNER B. M., FELDMAN B. E., CHENTANEZ N., O'BRIEN J. F.: Fluid animation with dynamic meshes. *ACM Trans. Graph. 25*, 3 (July 2006), 820–825. doi:10.1145/1141911.1141961. 115

[KLLR05] KIM B., LIU Y., LLAMAS I., ROSSIGNAC J.: Flowfixer: using BFECC for fluid simulation. In *Proceedings of the First Eurographics conference on Natural Phenomena* (Aire-la-Ville, Switzerland, Switzerland, 2005), NPH'05, Eurographics Association, pp. 51–56. doi:10.2312/NPH/NPH05/051-056. 107

[KM85] KIM J., MOIN P.: Application of a fractional-step method to incompressible Navier-Stokes equations. *Journal of Computational Physics 59*, 2 (1985), 308 – 323. doi:10.1016/0021-9991(85)90148-2. 104, 113

[KM90] KASS M., MILLER G.: Rapid, stable fluid dynamics for computer graphics. In *Proceedings of the 17th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1990), SIGGRAPH '90, ACM, pp. 49–57. doi:10.1145/97879.97884. 113

[KMB*09] KAUFMANN P., MARTIN S., BOTSCH M., GRINSPUN E., GROSS M.: Enrichment textures for detailed cutting of shells. *ACM Trans. Graph. 28*, 3 (2009), 50:1–50:10. doi:10.1145/1531326.1531356. 37

[KMBG08] KAUFMANN P., MARTIN S., BOTSCH M., GROSS M.: Flexible simulation of deformable models using discontinuous Galerkin FEM. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Aire-la-Ville, Switzerland, Switzerland, 2008), SCA '08, Eurographics Association, pp. 105–115. URL: http://dl.acm.org/citation.cfm?id=1632592.1632608. 37, 58

[KTJG08] KIM T., THÜREY N., JAMES D., GROSS M.: Wavelet turbulence for fluid simulation. *ACM Trans. Graph. 27*, 3 (Aug. 2008), 50:1–50:6. doi:10.1145/1360612.1360649. 116

[KTY09] KIKUUWE R., TABUCHI H., YAMAMOTO M.: An edge-based computationally efficient formulation of Saint Venant-Kirchhoff tetrahedral finite elements. *ACM Trans. Graph. 28*, 1 (Feb. 2009), 8:1–8:13. doi:10.1145/1477926.1477934. 39

[KW03] KRÜGER J., WESTERMANN R.: Linear algebra operators for GPU implementation of numerical algorithms. *ACM Trans. Graph. 22*, 3 (2003), 908–916. doi:10.1145/882262.882363. 40

[KYT*06] KHAREVYCH L., YANG W., TONG Y., KANSO E., MARSDEN J. E., SCHRÖDER P., DESBRUN M.: Geometric, variational integrators for computer animation. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Aire-la-Ville, Switzerland, Switzerland, 2006), SCA '06, Eurographics Association, pp. 43–51. URL:

http://dl.acm.org/citation.cfm?id=1218064.1218071. 67

[LAF11] LENTINE M., AANJANEYA M., FEDKIW R.: Mass and momentum conservation for fluid simulation. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (New York, NY, USA, 2011), SCA '11, ACM, pp. 91–100. doi:10.1145/2019406.2019419. 104

[LC87] LORENSEN W. E., CLINE H. E.: Marching cubes: A high resolution 3d surface construction algorithm. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA, 1987), SIGGRAPH '87, ACM, pp. 163–169. doi:10.1145/37401.37422. 110

[LGF04] LOSASSO F., GIBOU F., FEDKIW R.: Simulating water and smoke with an octree data structure. *ACM Trans. Graph. 23*, 3 (Aug. 2004), 457–462. doi:10.1145/1015706.1015745. 115

[Li98] LI Z.: A fast iterative algorithm for elliptic interface problems. *SIAM J. Numer. Anal. 35*, 1 (Feb. 1998), 230–254. doi:10.1137/S0036142995291329. 145

[LJWD08] LIU Y., JIAO S., WU W., DE S.: GPU accelerated fast FEM deformation simulation. In *APCCAS* (2008), pp. 606–609. doi:10.1109/APCCAS.2008.4746096. 41

[LLJ*11] LEVIN D. I. W., LITVEN J., JONES G. L., SUEDA S., PAI D. K.: Eulerian solid simulation with contact. *ACM Trans. Graph. 30*, 4 (July 2011), 36:1–36:10. doi:10.1145/2010324.1964931. 37

[LS07] LAI M.-J., SCHUMAKER L.: *Spline Functions on Triangulations*. Cambridge University Press, 2007. doi:10.1017/CBO9780511721588. 35, 43, 44, 58, 61

[LZF10] LENTINE M., ZHENG W., FEDKIW R.: A novel algorithm for incompressible flow using only a coarse grid projection. *ACM Trans. Graph. 29* (July 2010), 114:1–114:9. doi:10.1145/1778765.1778851. 116

[MCP*09] MULLEN P., CRANE K., PAVLOV D., TONG Y., DESBRUN M.: Energy-preserving integrators for fluid animation. *ACM Trans. Graph. 28*, 3 (July 2009), 38:1–38:8. doi:10.1145/1531326.1531344. 67, 116

[MDM*02] MÜLLER M., DORSEY J., MCMILLAN L., JAGNOW R., CUTLER B.: Stable real-time deformations. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation* (New York, NY, USA, 2002), SCA '02, ACM, pp. 49–54. doi:10.1145/545261.545269. 35

[MG04] MÜLLER M., GROSS M.: Interactive virtual materials. In *Proceedings of Graphics Interface 2004* (School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 2004), GI '04, Canadian Human-Computer Communications Society, pp. 239–246. URL: http://dl.acm.org/citation.cfm?id=1006058.1006087. 27, 35, 36, 46, 49, 84

[MKB*08] MARTIN S., KAUFMANN P., BOTSCH M., WICKE M., GROSS M.: Polyhedral finite elements using harmonic basis functions. In *Proceedings of the Symposium on Geometry Processing* (Aire-la-Ville, Switzerland, Switzerland, 2008), SGP '08, Eurographics Association, pp. 1521–1529. URL: http://dl.acm.org/citation.cfm?id=1731309.1731340. 37

[MLA10] MONAKOV A., LOKHMOTOV A., AVETISYAN A.: Automatically tuning sparse matrix-vector multiplication for GPU architectures. In *High Performance Embedded Architectures and Compilers*. Springer Berlin / Heidelberg, 2010. doi:10.1007/978-3-642-11515-8_10. 40, 79, 80

[MM13] MACKLIN M., MÜLLER M.: Position based fluids. *ACM Trans. Graph. 32*, 4 (July 2013), 104:1–104:12. doi:10.1145/2461912.2461984. 116

[MS06] MEZGER J., STRASSER W.: Interactive soft object simulation with quadratic finite elements. In *Proceedings of the 4th international conference on Articulated Motion and Deformable Objects* (Berlin, Heidelberg, 2006), AMDO'06, Springer-Verlag, pp. 434–443. doi:10.1007/11789239_45. 38, 42, 44, 56

[MSJT08] MÜLLER M., STAM J., JAMES D., THÜREY N.: Real time physics: Class notes. In *ACM SIGGRAPH 2008 Classes* (New York, NY, USA, 2008), SIGGRAPH '08, ACM, pp. 88:1–88:90. doi:10.1145/1401132.1401245. 34

[MST10] MCADAMS A., SIFAKIS E., TERAN J.: A parallel multigrid Poisson solver for fluids simulation on large grids. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Aire-la-Ville, Switzerland, Switzerland, 2010), SCA '10, Eurographics Association, pp. 65–74. URL: http://dl.acm.org/citation.cfm?id=1921427.1921438. 118, 133, 134, 135, 137, 138, 140, 141, 146

[MSW14] MICHELS D. L., SOBOTTKA G. A., WEBER A. G.: Exponential integrators for stiff elastodynamic problems. *ACM Trans. Graph. 33*, 1 (Feb. 2014), 7:1–7:20. doi:10.1145/2508462. 36

[MTPS08] MEZGER J., THOMASZEWSKI B., PABST S., STRASSER W.: Interactive physically-based shape editing. In *Proceedings of the 2008 ACM symposium on Solid and physical modeling* (New York, NY, USA, 2008), SPM '08, ACM, pp. 79–89. doi:10.1145/1364901.1364915. 31, 38, 42, 43, 44, 46, 47, 52, 53, 93, 94

[Mül09] MÜLLER M.: Fast and robust tracking of fluid surfaces. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (New York, NY, USA, 2009), SCA '09, ACM, pp. 237–245. doi:10.1145/1599470.1599501. 114

[MZS*11] MCADAMS A., ZHU Y., SELLE A., EMPEY M., TAMSTORF R., TERAN J., SIFAKIS E.: Efficient elasticity for character skinning with contact and collisions. *ACM Trans. Graph. 30*, 4 (July 2011), 37:1–37:12. doi:10.1145/2010324.1964932. 36, 37

[NKJF09] NESME M., KRY P. G., JERÁBKOVÁ L., FAURE F.: Preserving topology and elasticity for embedded deformable models. *ACM Trans. Graph. 28*, 3 (July 2009), 52:1–52:9. doi:10.1145/1531326.1531358. 37

[NMG09] NG Y. T., MIN C., GIBOU F.: An efficient fluid-solid coupling algorithm for single-phase flows. *J. Comput. Phys. 228*, 23 (Dec. 2009), 8807–8829. doi:10.1016/j.jcp.2009.08.032. 117, 118, 122, 134, 135, 145, 146

[NMK*06] NEALEN A., MÜLLER M., KEISER R., BOXERMAN E., CARLSON M.: Physically based deformable models in computer graphics. *Computer Graphics Forum 25*, 4 (2006), 809–836. doi:10.1111/j.1467-8659.2006.01000.x. 34, 112

[NMP*05] NESME M., MARCHAL M., PROMAYON E., CHABANAS M., PAYAN Y., FAURE F., ET AL.: Physically realistic interactive simulation for biological soft tissues. *Recent Research Developments in Biomechanics 2* (2005), 1–22. 38

[NPF05] NESME M., PAYAN Y., FAURE F.: Efficient, physically plausible finite elements. In *Eurographics (short papers)* (Dublin, Irlande, August 2005), Dingliana J., Ganovelli F., (Eds.). doi:10.2312/egs.20051028. 39, 66

[NVI11] NVIDIA: NVIDIA CUDA sparse matrix library, 2011. URL: http://developer.nvidia.com/cuSPARSE. 40

[NVI15] NVIDIA: NVIDIA CUDA Compute Unified Device Architecture, 2015. [online, accessed on Sep. 24th 2015]. URL: http://www.nvidia.com/object/cuda_home_new.html. 40, 75, 76, 77, 124

[OF02] OSHER S., FEDKIW R.: *Level Set Methods and Dynamic Implicit Surfaces*. Springer Verlag, New York, 2002. doi:10.1007/b98879. 110

[OH99] O'BRIEN J. F., HODGINS J. K.: Graphical modeling and animation of brittle fracture. In *SIGGRAPH 1999* (1999), pp. 137–146. doi:10.1145/311535.311550. 30

[OS88] OSHER S., SETHIAN J. A.: Fronts propagating with curvature-dependent speed: Algorithms based on hamilton-jacobi formulations. *Journal of Computational Physics 79*, 1 (1988), 12 – 49. doi:10.1016/0021-9991(88)90002-2. 114

[OSV11] OBERHUBER T., SUZUKI A., VACATA J.: New row-grouped CSR format for storing the sparse matrices on GPU with implementation in CUDA. *Acta Technica 4* (2011), 447–466. 40, 77, 81

[PDA00] PICINBONO G., DELINGETTE H., AYACHE N.: Real-time large displacement elasticity for surgery simulation: Non-linear tensor-mass model. In *Medical Image Computing and Computer-Assisted Intervention - MICCAI 2000*, Delp S., DiGoia A., Jaramaz B., (Eds.), vol. 1935 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2000, pp. 643–652. doi:10.1007/978-3-540-40899-4_66. 35

[PnSdSSM09] PEÑA SERNA S., DA SILVA J. G. B. R., STORK A., MARCOS A. F.: Neighboring-based linear system for dynamic meshes. In *Workshop in Virtual Reality Interactions and Physical Simulation* (2009), The Eurographics Association, pp. 95–103. doi:10.2312/PE/vriphys/vriphys09/095-103. 39, 47, 48

[PO09] PARKER E. G., O'BRIEN J. F.: Real-time deformation and fracture in a game environment. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (New York, NY, USA, 2009), SCA '09, ACM, pp. 165–175. doi:10.1145/1599470.1599492. 35, 43, 46, 47, 50

[PTSG09] PFAFF T., THUEREY N., SELLE A., GROSS M.: Synthetic turbulence using artificial boundary layers. In *ACM SIGGRAPH Asia 2009 papers* (New York, NY, USA, 2009), SIGGRAPH Asia '09, ACM, pp. 121:1–121:10. doi:10.1145/1661412.1618467. 116

[RGTC98] ROTH S., GROSS M., TURELLO S., CARLS F.: A Bernstein-Bézier based approach to soft tissue simulation. *Computer Graphics Forum 17*, 3 (1998), 285–294. doi:10.1111/1467-8659.00275. 38, 42, 43

[Rot02] ROTH S. H. M.: *Bernstein–Bézier Representations for Facial Surgery Simulation*. PhD thesis, Swiss Federal Institute of Technology, 2002. 42, 43

[SA07a] SORKINE O., ALEXA M.: As-rigid-as-possible surface modeling. In *Proceedings of the Fifth Eurographics Symposium on Geometry Processing* (Aire-la-Ville, Switzerland, Switzerland, 2007), SGP '07, Eurographics Association, pp. 109–116. URL: http://dl.acm.org/citation.cfm?id=1281991.1282006. 10

[SA07b] SPURK J. H., AKSEL N.: *Strömungslehre - Einführung in die Theorie der Strömungen*. Springer, Berlin [u.a.], 2007. doi:10.1007/978-3-642-13143-1. 99

[SABS14] SETALURI R., AANJANEYA M., BAUER S., SIFAKIS E.: SPGrid: A sparse paged grid structure applied to adaptive smoke simulation. *ACM Trans. Graph. 33*, 6 (Nov. 2014), 205:1–205:12.

doi:10.1145/2661229.2661269. 118

[SB12] SIFAKIS E., BARBIC J.: FEM simulation of 3D deformable solids: A practitioner's guide to theory, discretization and model reduction. In *ACM SIGGRAPH 2012 Courses* (2012), ACM, pp. 20:1–20:50. doi:10.1145/2343483.2343501. 34

[SBCL06] SCHWARTZ P., BARAD M., COLELLA P., LIGOCKI T.: A cartesian grid embedded boundary method for the heat equation and Poisson's equation in three dimensions. *Journal of Computational Physics 211*, 2 (2006), 531 – 550. doi:10.1016/j.jcp.2005.06.010. 117, 118

[SBH09] SIN F., BARGTEIL A. W., HODGINS J. K.: A point-based method for animating incompressible flow. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (New York, NY, USA, 2009), SCA '09, ACM, pp. 247–255. doi:10.1145/1599470.1599502. 116

[SFK*08] SELLE A., FEDKIW R., KIM B., LIU Y., ROSSIGNAC J.: An unconditionally stable MacCormack method. *Journal of Scientific Computing 35*, 2 (2008), 350–371. doi:10.1007/s10915-007-9166-4. 107

[SG04] SCHENK O., GÄRTNER K.: Solving unsymmetric sparse systems of linear equations with PARDISO. *Future Generation Computer Systems 20*, 3 (2004), 475–487. doi:10.1016/j.future.2003.07.011. 31

[SGS10] STONE J. E., GOHARA D., SHI G.: Opencl: A parallel programming standard for heterogeneous computing systems. *IEEE Des. Test 12*, 3 (May 2010), 66–73. doi:10.1109/MCSE.2010.69. 75

[She94] SHEWCHUK J. R.: *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain*. Tech. rep., Carnegie Mellon University, Pittsburgh, PA, USA, 1994. 32, 81, 82, 129

[SHST12] STOMAKHIN A., HOWES R., SCHROEDER C., TERAN J. M.: Energetically consistent invertible elasticity. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Aire-la-Ville, Switzerland, Switzerland, 2012), SCA '12, Eurographics Association, pp. 25–32. URL: http://dl.acm.org/citation.cfm?id=2422356.2422361. 36

[Si15] SI H.: Tetgen, a delaunay-based quality tetrahedral mesh generator. *ACM Trans. Math. Softw. 41*, 2 (Feb. 2015), 11:1–11:36. doi:10.1145/2629697. 21

[SRF05] SELLE A., RASMUSSEN N., FEDKIW R.: A vortex particle method for smoke, water and explosions. *ACM Trans. Graph. 24*, 3 (July 2005), 910–914. doi:10.1145/1073204.1073282. 116

[SSB13] SIN F. S., SCHROEDER D., BARBIC J.: Vega: Non-linear FEM deformable object simulator. *Computer Graphics Forum 32*, 1 (2013), 36–48. doi:10.1111/j.1467-8659.2012.03230.x. 36, 38

[SSW*12] STORK A., SEVILMIS N., WEBER D., GORECKY D., STAHL C., LOSKYLL M., MICHEL F.: Enabling virtual assembly training in and beyond the automotive industry. In *Proceedings of the VSMM 2012* (2012), International Society on Virtual Systems and MultiMedia, IEEE Computer Society, Los Alamitos, Calif., pp. 347–352. doi:10.1109/VSMM.2012.6365944. 165

[SSX06] SHU S., SUN D., XU J.: An algebraic multigrid method for higher-order finite element discretizations. *Computing 77*, 4 (2006), 347–377. doi:10.1007/s00607-006-0162-6. 39

[ST08] SCHMEDDING R., TESCHNER M.: Inversion handling for stable deformable modeling. *Vis. Comput. 24*, 7 (July 2008), 625–633. doi:10.1007/s00371-008-0243-y. 29, 35, 36

[Sta99] STAM J.: Stable fluids. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1999), SIGGRAPH '99, ACM Press/Addison-Wesley Publishing Co., pp. 121–128. doi:10.1145/311535.311548. 104, 106, 113, 133

[TBHF03] TERAN J., BLEMKER S., HING V. N. T., FEDKIW R.: Finite volume methods for the simulation of skeletal muscle. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation* (Aire-la-Ville, Switzerland, Switzerland, 2003), SCA '03, Eurographics Association, pp. 68–74. URL: http://dl.acm.org/citation.cfm?id=846276.846285. 37

[TCO08] TAYLOR Z., CHENG M., OURSELIN S.: High-speed nonlinear finite element analysis for surgical simulation using graphics processing units. *Medical Imaging, IEEE Transactions on 27*, 5 (May 2008), 650–663. doi:10.1109/TMI.2007.913112. 41

[TPBF87] TERZOPOULOS D., PLATT J., BARR A., FLEISCHER K.: Elastically deformable models. *SIGGRAPH Comput. Graph. 21*, 4 (Aug. 1987), 205–214. doi:10.1145/37402.37427. 35

[TS01] TROTTENBERG U., SCHULLER A.: *Multigrid*. Academic Press, Inc., Orlando, FL, USA, 2001. 39, 60, 118, 137

[TSIF05] TERAN J., SIFAKIS E., IRVING G., FEDKIW R.: Robust quasistatic finite elements and flesh simulation. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation* (New York, NY, USA, 2005), SCA '05, ACM, pp. 181–190. doi:10.1145/1073368.1073394. 38

[VOFG10] VAZQUEZ F., ORTEGA G., FERNÁNDEZ J.-J., GARZÓN E. M.: Improving the performance of the sparse matrix vector product with GPUs. In *CIT'10* (2010), pp. 1146–1151. doi:10.1109/CIT.2010.208. 40, 80, 90

[WBG07] WICKE M., BOTSCH M., GROSS M.: A finite element method on convex polyhedra. *Computer Graphics Forum 26*, 3 (2007), 355–364. doi:10.1111/j.1467-8659.2007.01058.x. 37

[WBS*13] WEBER D., BENDER J., SCHNOES M., STORK A., FELLNER D. W.: Efficient GPU data structures and methods to solve sparse linear systems in dynamics applications. *Computer Graphics Forum 32*, 1 (2013), 16–26. doi:10.1111/j.1467-8659.2012.03227.x. 5, 38, 66, 74, 93, 165

[Web08] WEBER D.: Trivariate Bernstein-Bézier-Techniken für Finite Elemente zur interaktiven Simulation von Deformationen, 2008. Darmstadt, TU, Diplomarbeit, 2008. 5, 10, 42

[WH04] WU W., HENG P. A.: A hybrid condensed finite element model with GPU acceleration for interactive 3d soft tissue cutting. *Computer Animation and Virtual Worlds 15*, 3-4 (2004), 219–227. doi:10.1002/cav.24. 41

[WKS*11] WEBER D., KALBE T., STORK A., FELLNER D. W., GOESELE M.: Interactive deformable models with quadratic bases in Bernstein-Bézier-form. *The Visual Computer 27* (2011), 473–483. doi:10.1007/s00371-011-0579-6. 5, 38, 42, 93, 165

[WL04] WAN J. W. L., LIU X.-D.: A boundary condition–capturing multigrid approach to irregular boundary problems. *SIAM J. Sci. Comput. 25*, 6 (June 2004), 1982–2003. doi:10.1137/S1064827503428540. 118

[WMFB11] WOJTAN C., MÜLLER-FISCHER M., BROCHU T.: Liquid simulation with mesh-based surface tracking. In *ACM SIGGRAPH 2011 Courses* (New York, NY, USA, 2011), SIGGRAPH '11, ACM, pp. 8:1–8:84. doi:10.1145/2037636.2037644. 114

[WMRA*14] WEBER D., MUELLER-ROEMER J., ALTENHOFEN C., STORK A., FELLNER D. W.: A p-multigrid algorithm using cubic finite elements for efficient deformation simulation. In *Virtual Reality Interactions and Physical Simulations (VRIPhys)* (2014), pp. 49–58. doi:10.2312/vriphys.20141223. 5, 56, 93, 165

[WMRA*15] WEBER D., MUELLER-ROEMER J., ALTENHOFEN C., STORK A., FELLNER D. W.: Deformation simulation using cubic finite elements and efficient *p*-multigrid methods. *Computers & Graphics 53, Part B* (2015), 185 – 195. doi:10.1016/j.cag.2015.06.010. 5, 56, 93, 147, 165

[WMRSF15] WEBER D., MUELLER-ROEMER J., STORK A., FELLNER D. W.: A cut-cell geometric multigrid poisson solver for fluid simulation. *Computer Graphics Forum (Eurographics proceedings) 34*, 2 (2015), 481–491. doi:10.1111/cgf.12577. 5, 133, 165

[WP10] WEISSMANN S., PINKALL U.: Filament-based smoke with vortex shedding and variational reconnection. *ACM Trans. Graph. 29*, 4 (July 2010), 115:1–115:12. doi:10.1145/1778765.1778852. 116

[WPnSSF11] WEBER D., PEÑA SERNA S., STORK A., FELLNER D. W.: Rapid CFD for the early conceptual design phase. In *The Integration of CFD into the Product Development Process* (2011), International Association for the Engineering Analysis Community (NAFEMS), NAFEMS, Glasgow, p. 9. 5, 121, 147, 165

[WTF06] WEINSTEIN R. L., TERAN J., FEDKIW R.: Dynamic simulation of articulated rigid bodies with contact and collision. In *IEEE Transactions on Visualization and CG* (2006), vol. 12, pp. 365–374. doi:10.1109/TVCG.2006.48. 85

[WTGT09] WOJTAN C., THÜREY N., GROSS M., TURK G.: Deforming meshes that split and merge. *ACM Trans. Graph. 28*, 3 (July 2009), 76:1–76:10. doi:10.1145/1531326.1531382. 114

[ZB05] ZHU Y., BRIDSON R.: Animating sand as a fluid. *ACM Trans. Graph. 24*, 3 (July 2005), 965–972. doi:10.1145/1073204.1073298. 104, 113

[ZB13] ZHAO Y., BARBIC J.: Interactive authoring of simulation-ready plants. *ACM Trans. Graph. 32*, 4 (July 2013), 84:1–84:12. doi:10.1145/2461912.2461961. 37

[ZLC*13] ZHU B., LU W., CONG M., KIM B., FEDKIW R.: A new grid structure for domain extension. *ACM Trans. Graph. 32*, 4 (jul 2013), 63:1–63:12. doi:10.1145/2461912.2461999. 116

[ZQC*14] ZHU B., QUIGLEY E., CONG M., SOLOMON J., FEDKIW R.: Codimensional surface tension flow on simplicial complexes. *ACM Trans. Graph. 33*, 4 (July 2014), 111:1–111:11. doi:10.1145/2601097.2601201. 115

[ZSTB10] ZHU Y., SIFAKIS E., TERAN J., BRANDT A.: An efficient multigrid method for the simulation of high-resolution elastic solids. *ACM Trans. Graph. 29*, 2 (Apr. 2010), 16:1–16:18. doi:10.1145/1731047.1731054. 39

[ZT00] ZIENCKIEWICZ O. C., TAYLOR R. L.: *The Finite Element Method*. Butterworth-Heinemann, Oxford, 2000. 19, 21, 34, 38